ESCUELA TÉCNICA SUPERIOR DE INGENIEROS INDUSTRIALES Y DE TELECOMUNICACIÓN

Titulación:

INGENIERO TÉCNICO EN INFORMÁTICA DE GESTIÓN

Título del proyecto:

DESARROLLO DE UN CONTROLADOR DIFUSO PARA EL ROBOT KHEPERA II

Alumnas: Maitane Larrayoz Olaetxea y Amaia Martínez Echeverría

Tutor: Miguel Pagola Barrio

Pamplona, 29 de abril de 2010

Índice

Capítulo 1. Introducción

1.1. Introducción y definición del problema

Se comienza este proyecto con el estudio del robot Khepera II el cual será la base del mismo. Este estudio supondrá la lectura y entendimiento completo del manual de usuario, previamente traducido por un alumno que realizó otro proyecto con este robot, y de la memoria redactada por éste. Con este estudio se pretende comprender el funcionamiento del robot (sus modos de operación, estructura mecánica, comunicación con un ordenador…). Una vez se haya terminado esta etapa del proyecto, se procederá a realizar una serie de pruebas mediante las cuales comprobaremos el correcto comportamiento del robot. Cabe destacar, que las pruebas realizadas son las propuestas por el mismo alumno citado anteriormente, por lo que no las detallaremos en estas memorias. En realidad, el objetivo de esta etapa será obtener una cierta soltura en el uso y la programación del robot a través de un ordenador.

Fig. 1 Robot Khepera II.

En este momento debemos ser capaces de desarrollar sencillos programas para la interacción con el robot, así pues, para no dañar el hardware del Khepera II, antes de comunicarnos con él directamente probaremos dichos programas con el simulador Kiks (versión 2.2.0). Para ello, estudiaremos el funcionamiento del simulador y nos familiarizaremos con él, ya que nos será útil en las primeras etapas de la creación de los controladores.

A continuación, una vez familiarizados con el robot Khepera II y el simulador Kiks, estudiaremos algunos conceptos básicos de la materia lógica difusa, ya que ésta se imparte en el segundo ciclo de informática y por lo tanto, nos es desconocida. Una vez hecho esto, nos pondremos a construir el controlador difuso aplicando los conocimientos adquiridos mediante dicho estudio. En este apartado desarrollaremos una serie de “comportamientos” para distintas situaciones en las que puede encontrarse el robot. Estos

comportamientos si que irán explicados y detallados en la memoria, así como las diferentes pruebas y resultados relacionados con los mismos.

Los comportamientos se realizarán, en un principio, a través del software Matlab, que se comunicará con el Khepera II gracias a un conjunto de librerías proporcionadas por *K-Team Corporation.*

Otra de las etapas a la hora de desarrollar el proyecto, será la familiarización con otro simulador, en este caso el simulador Webots, ya que será con éste con el que realicemos tanto los pasos finales en la etapa de implementación del controlador difuso, como las posteriores simulaciones para la optimización del mismo.

Llegados a este punto, optimizaremos el controlador mediante un algoritmo genético (GA), para lo cual estudiaremos los distintos tipos de GA's existentes así como el funcionamiento de los mismos. Así pues, uno de los apartados clave de esta etapa será el diseño del algoritmo genético y su adaptación a nuestro problema.

En estas memorias se explicará detalladamente tanto el desarrollo de comportamientos para el robot móvil, como los algoritmos utilizados, resultados obtenidos, breves introducciones a los temas desconocidos (lógica difusa, GA's…), etc.

El objetivo principal del proyecto consiste en dotar al robot Khepera II de la capacidad de moverse de manera autónoma sin colisionar mediante la interacción con el mundo real a través de la información captada por sus sensores infrarrojos de proximidad.

El material del que se dispone para este proyecto consiste en el paquete básico del robot Khepera II, en el que se incluyen el robot, toma de corriente, cable de carga de las baterías, cable para comunicarse con el puerto serie, fundas de repuesto para las ruedas y una torreta para la conexión inalámbrica entre el robot y el ordenador.

1.2. Introducción a la robótica móvil

La robótica es una disciplina que combina todas aquellas actividades relacionados con el estudio, diseño, construcción, operación y manutención de robots. Es un campo de trabajo que combina diferentes disciplinas como Ingeniería Eléctrica, Ingeniería Electrónica, Ingeniería Mecánica, Ciencias de la Computación, Matemáticas, Física, Biología, Neurociencias, etc. A su vez, la robótica es sinónimo de progreso y desarrollo tecnológico; los países y las empresas que cuentan con una fuerte presencia de robots no solamente consiguen una extraordinario competitividad y productividad, sino que transmiten una imagen de modernidad. En los países desarrollados, las inversiones en tecnologías robóticas han crecido de forma significativa y muy por encima de otros sectores.

Desde siempre el hombre ha tenido el deseo de poseer el conocimiento del propio de su ser, de su comportamiento y el de otras especies, a fin de crear agentes con la capacidad autónoma con los cuales pueda compartir la inteligencia, para encargarle la realización de tareas que a el le desagradan por ser monótonas, complicadas y peligrosas.

La robótica móvil es un área que se encuentra en continuo proceso de investigación, por ello, muchas universidades poseen laboratorios que se centran ella. Los robots móviles son aquellos capaces de realizar movimientos en un entorno dado, es decir, no están fijados a un entorno físico concreto.

La forma en la que un robot móvil puede navegar por una superficie sólida es variada, los más comunes son las ruedas, cadenas y las patas. Además. Suelen poseer sistemas sensoriales con sensores de distancia o alcance, sensores de luz y fotorresistencia.

Por otro lado, estos robots utilizan diferentes tipos de navegación. A continuación, se detallan algunos de estos tipos:

- **Teledirigido**. Se controla mediante un dispositivo externo, como un joystick.

- **Teledirigido vigilante**. Se controla mediante un dispositivo externo, como un joystick, pero posee la habilidad de detectar y evitar obstáculos.

- **Robot que sigue la línea**. Los vehículos mas recientes guiados automáticamente siguen este tipo de navegación. Siguen una línea visual, manteniéndola en el centro de los sensores. Este tipo de navegación no evade los obstáculos, simplemente para cuando se encuentra con uno en la línea.

- **De manera automática al azar**: Utilizan el movimiento al azar que básicamente rebotan en las paredes. El robot conoce la localización y como alcanzar varias metas.

- **Guiado de manera automática**.

La robótica ha sido una de las tecnologías industriales fundamentales durante más de 30 años, contribuyendo en gran medida al aumento de la productividad en muchos sectores. Sólo en época muy reciente se puede afirmar que por fin los robots están empezando a salir de las fábricas para introducirse en nuestras casas, oficinas, hospitales, museos u otros espacios públicos.

Entre las aplicaciones más innovadoras de la robótica y la automatización caben destacar algunos sectores como:

Aplicaciones industriales

La Robótica y la Automatización siempre han ofrecido al sector industrial un excelente compromiso entre productividad y flexibilidad, una calidad uniforme de los productos, una sistematización de los procesos y la posibilidad de supervisar y/o controlar las plantas según diferentes parámetros y criterios. Se pueden destacar cuatro ventajas principales de los sistemas robotizados:

- Productividad: La introducción de robots en operaciones tales como soldadura, manipulación de productos, pintura, ensamblado, almacenaje o control de calidad, permite reducir sustancialmente el tiempo de las operaciones, aumentando la productividad y reduciendo los costes.

- Flexibilidad: Los sistemas robotizados actuales destacan por ser máquinas y sistemas flexibles que pueden adaptarse a la fabricación de una familia de productos sin la necesidad de que se modifique o se detenga la cadena de producción.

- Calidad: La repetición de las tareas realizadas por los robots y el control de productividad en todos los ámbitos de la factoría permiten asegurar un alto nivel de calidad del producto final. Además, los sistemas robotizados no solamente se emplean para la fabricación, sino también para medir la calidad del producto final mediante sistemas mecánicos (palpadores) u ópticos (láser).

- Seguridad laboral: La introducción de robots en operaciones tales como soldadura y pintura o manipulados de sustancias peligrosas o de materiales a altas temperaturas permiten la eliminación de tareas laborales penosas y la disminución de accidentes laborales.

Como resultado, la robotización permite mejorar la calidad y las condiciones de trabajo, sustituyendo tareas penosas por otras que se efectúan en condiciones mucho más ventajosas.

Transporte de mercancías

Sin lugar a dudas, uno de los ámbitos más importantes de los robots móviles en aplicaciones industriales es el transporte de mercancías. Entre éstos se encuentran los vehículos autoguiados (AVG), los AVG están diseñados para transportar grandes cargas, como pueden ser las transpaletas apiladoras.

Actualmente, se ha dado un salto más en los AGV, creando los sistemas de transporte de material automático (AMTS). Estos sistemas solventan la incapacidad de los vehículos autoguiados para percibir su entorno y reduce la dependencia humana en ellos. Este sistema permite que los robots se desplacen por una fábrica sin necesidad de ningún tipo

de guía. Estos vehículos están equipados con cámaras y visión por barridos sucesivos de un láser utilizados en tareas de navegación y control.

Fig. 2 Vehículos autoguiados.

Agricultura

Podría decirse que no se están desarrollando nuevos robots con aplicaciones agrícolas, sino que toda la maquinaria ya existente está recibiendo las innovaciones tecnológicas que aporta la robótica móvil. Los avances en este sector no sólo están encaminados a la automatización de las labores agrícolas, también buscan incrementar la seguridad de los operarios mediante el uso de sistemas de percepción y posicionamiento. Por ejemplo, los tractores y cosechadoras son como los que podemos ver en cualquier tierra de labor, con la peculiaridad de que no tienen conductor.

Fig. 3 Tractor y cosechadora autónomos

El sector automovilístico

Otra de las áreas más importantes de la robótica móvil es el sector automovilístico. La robótica está completamente integrada en la fabricación, pero el hecho de construir automóviles capaces de andar sin

conductor está en continuo proceso de investigación. Estos automóviles deberán poseer la capacidad de frenar, acelerar y elegir opciones sin la necesidad de un humano al volante.

La tarea de lograr un automóvil robot eficiente que pueda circular por un centro urbano o cualquier otro entorno es complicada, ya que existen muchos factores a tener en cuenta. Estos factores pueden ser variados, pero entre ellos se encuentran los otros coches, las personas y los semáforos, así como la complejidad de estacionar en un centro urbano. Además de ello, hay que tomar en cuenta la reacción que podría tener el automóvil con un imprevisto tal como un fallo en el sistema. En este caso el peligro sería muy grave.

Actualmente se están desarrollando los primeros automóviles sin conductor, de hecho, existen desafíos para este tipo de automóviles. Aunque todavía queda mucho terreno que explorar, muchos ven abierta la posibilidad a futuro de que la tecnología pueda eventualmente utilizarse para automóviles de calle que se conduzcan solos.

Fig. 4 Vehículo autónomo.

Aplicaciones innovadoras y de servicio

Los robots de servicios son aquellos que de forma semiautomática o automática realizan servicios en beneficio de los humanos o para el mantenimiento de infraestructuras y equipos, excluidas las operaciones de fabricación.

El sector servicios, tanto personal como colectivo, es una de las áreas de aplicación más novedosa. Se estima que en los próximos diez años el sector pueda requerir necesidades en robótica con un volumen de negocio comparable con el del sector industrial.

En este sector, existen algunos desarrollos sorprendentes para ayudar en el cuidado de las personas discapacitadas mayores, como por ejemplo una silla de ruedas servo-controlada desde un computador que incluye un brazo muy ligero, capaz de proporcionar al usuario una gran movilidad a la vez que le permite realizar tareas como abrir la puerta o lavarse los dientes. Otro ejemplo más avanzado son los robots

“escaladores” que no solamente se mueven conjuntamente con la silla de ruedas, sino que también pueden moverse de manera independiente por el entorno doméstico.

La irrupción de la automatización en los servicios y el ocio permite mejorar la calidad de vida de los ciudadanos.

Inspección y vigilancia

Una de las aplicaciones más habituales son los robots móviles aplicados en tareas de inspección y vigilancia. Este tipo de robots suelen incorporar cámaras de video-vigilancia, de infrarrojos, detectores de ruidos y gases y otros sistemas específicos.

Actualmente, ya se están aplicando estos sistemas con éxito en tareas de vigilancia de grandes superficies comerciales, almacenes de logística, laboratorios, edificios de oficinas, estadios de fútbol, fábricas de automóviles o incluso en domicilios particulares, entre otras, en cuanto a espacios de interior se refiere. Aunque también se estas utilizando robots de vigilancia para exteriores.

Estos sistemas de vigilancia, inspección y seguridad incorporan las más recientes tecnologías en sistemas de comunicación, navegación y control. Y de este modo, nos permiten también optimizar las tareas de vigilancia y, por tanto, la seguridad.

Por otro lado existen robots capaces de inspeccionar zonas peligrosas para las personas como pueden ser incendios, escombros, etc. en busca de supervivientes. Así como, para inspeccionar conductos.

Fig. 5 Tareas de inspección de conductos y tareas de rescate en catástrofes, respectivamente.

Uso doméstico y oficina

Las aplicaciones de robots móviles en el ámbito doméstico y de oficina, es un largo camino largo hacia resultados perfectos, sin embargo,

estamos empezando a ver la aparición de robots no excesivamente inteligentes, pero si eficientes para ciertas tareas concretas.

Actualmente, se pueden comprar robots que realizan tareas sencillas, como pueden ser la de pasar la aspiradora, cortar el césped, entretener a los niños o realizar pequeñas tareas de servicio como traer café o aperitivos. Esta perspectiva parece indicar que en un futuro podría aparecer lo que podríamos denominar el 'robot personal', un robot doméstico o de oficina de propósito general. De momento se prevé que el uso doméstico de los robots se multiplicará por siete en los próximos años.

Fig. 6 Robot aspiradora VC-RP30W de Samsung

Ocio y entretenimiento

Uno de los campos con mayor potencial de crecimiento en el sector de robots de servicios es el de la educación y el entretenimiento. Sus aplicaciones potenciales son enormes: cuidado y vigilancia de niños, revisión de deberes escolares, juegos educativos, juego con 'mascota', etc.

El ejemplo más destacado de este tipo de robots son las diferentes mascotas: perros, gatos, loros. Las ventas de estos robots han sobrepasado todas las expectativas.

Fig. 7 Robot Aibo y robot Qrio de Sony.

Servicios de comunidad

En el apartado de robots de servicios a infraestructuras y equipos, la aplicación de limpieza es una de las más demandadas. La limpieza de grandes superficies interiores se efectúa con robots móviles autónomos equipados con las herramientas de limpieza necesarias y con el mapa del local.

Otro ámbito de aplicación de gran relevancia, en el cual se continúa trabajando en el desarrollo de nuevas técnicas de interacción hombre-máquina, es el de los robots destinados a actuar como guías de museos o exposiciones o incluso como recepcionistas. A parte de las técnicas propias de navegación y localización de este tipo de robots, se esta comenzando a dotarles de ciertas características que les permitan interaccionar y comunicarse efectivamente con las personas. Para ello, se han implementado mecanismos de localización de personas.

Fig. 8 RoboX, el robot guía de la Swiss National Exhibition, Expo.

1.3. Objetivos generales

- Desarrollo de un programa que implemente o incorpore una serie de comportamientos enfocados al movimiento autónomo del robot *Khepera II*.

- Optimización del programa o algoritmo anterior mediante el uso de un algoritmo genético.

- Construcción de un circuito por el cual el robot *Khepera II* deberá ser capaz de moverse por sí mismo sin colisionar.

1.4. Objetivos específicos

- Creación de diferentes comportamientos para el robot (pequeños controladores difusos).

- Comprensión y estudio de la lógica difusa y los sistemas basados en ella.

- Desarrollo de un programa principal que implemente o incorpore los anteriores comportamientos enfocado al movimiento autónomo del robot *Khepera II*.

- Estudio de algoritmos genéticos y desarrollo del adecuado para nuestro problema.

- Optimización de cada uno de los comportamientos mediante el uso de nuestro algoritmo genético.

- Construcción de un circuito por el cual el robot *Khepera II* deberá ser capaz de moverse por sí mismo sin colisionar.

1.5. Hardware y Software a utilizar

HARDWARE	SOFTWARE
Kit Khepera II PC Cable RS232	Windows XP, Microsoft. KTProject, K-Team S.A. Tera Term Pro, K-Team S.A. Kiks 2.2.0 Webots 6.2.1 (*versión de evaluación*) Matlab, MathWorks.

Tabla 1

Capítulo 2. Controlador difuso para la navegación de un robot móvil

2.1. Introducción

El objetivo de este capítulo es crear diferentes comportamientos de navegación como lo son el seguimiento de pared, avanzar por pasillo y entrar por puerta. Estos comportamientos se diseñarán mediante controladores basados en lógica difusa. Una vez diseñados los comportamientos por separado, al final de este capitulo se proseguirá a incorporar los mismos a un programa, el cual será el controlador difuso propiamente dicho, que consistirá en la realización de un recorrido.

Para la realización de estos controladores difusos es imprescindible el estudio previo de los diferentes aspectos de la lógica difusa, también llamada lógica borrosa, y la estructura de dichos controladores. Así pues, antes de diseñar los controladores intentaremos familiarizarnos con el área de la inteligencia artificial que abarca la lógica difusa.

Además, estos controladores deberán ser diseñados de la forma más eficaz posible para lograr controladores que ejecuten el comportamiento deseado de una manera aceptable y que permitan su posterior optimización en otros capítulos.

2.2. Conceptos básicos sobre lógica difusa

La lógica borrosa o difusa permite tratar información imprecisa en términos de conjuntos difusos. Dicha lógica en vez de trabajar con el clásico concepto de inclusión o exclusión, introduce una función que especifica el **grado de pertenencia** de una variable al concepto dado por la etiqueta que le da nombre ha dicho conjunto tomando valores entre 0 y 1. En cambio, en términos de la lógica clásica una proposición tomará únicamente un valor lógico 0 ó 1, en definitiva, verdadero o falso.

Poniendo un ejemplo, un objeto a una distancia de 50 metros; en términos de lógica clásica estará lejos, en cambio, en términos de lógica difusa podrá estar un poco lejos o más o menos a una distancia intermedia. Siguiendo con el ejemplo, si el objeto se encuentra a 49 metros; en términos de lógica clásica estará cerca, mientras que en lógica difusa podrá estar un poco cerca o bastante cerca.

Luego, la lógica borrosa, en contra posición de la lógica clásica, se adapta mejor al mundo real, e incluso es capaz comprender los cuantificadores de nuestras lenguaje del tipo “a medias”, “bastante”, ”casi”, ”un poco”, “mucho”... El éxito de utilizar dicha lógica radica en el hecho de que no tiene

sentido buscar la solución a un problema que no está perfectamente definido por medio de un planteamiento matemático muy exacto, cuando es el ser humano el primero que razona empleando la inexactitud.

Basándonos en el ejemplo mencionado anteriormente, introduciremos ciertos términos importantes dentro del área de la lógica difusa. Una **variable lingüística** es aquella que puede tomar por valor términos del lenguaje natural, que son las palabras que desempeñan el papel de **etiquetas** en un conjunto borroso. En nuestro ejemplo, distancia será una variable lingüística que puede tomar diferentes etiquetas como lo son lejos, intermedia, cerca… No obstante, a una variable lingüística también podrán asignarse variable numéricos, como es “50 metros”.

Una vez introducidos diferentes aspectos de la lógica borrosa como las variables lingüísticas y las etiquetas, se proseguirá con **la función de inclusión o pertenencia de un conjunto borroso** (membership function). Dicha función consiste en un conjunto de pares ordenados $F=\{(u, \mu_F(u)) / u \in U\}$, donde $\mu_F(u)$ representa el grado en el que $u \in U$ está incluida en el concepto que representa la etiqueta F. Para la definición de estas pertenencias se utilizan convencionalmente ciertas familias de formas estándar. Las más frecuentes son las de tipo trapezoidal, triangular, S y tipo π (gaussiana), pero no obstante existen otras formas. En la figura que se muestra a continuación se puede observar algunas.

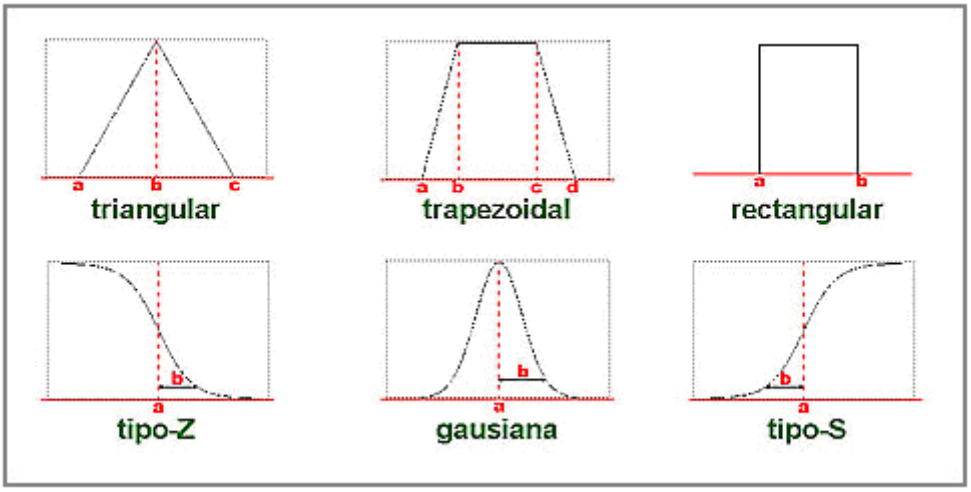


Fig. 9

Por otro lado cabe destacar que un aspecto clave de la lógica difusa es **el solapamiento entre los diferentes conjuntos**, es decir, dos conjuntos estarán solapados si su intersección es no nula. De esta forma el vaso podrá estar medio vacío, o medio lleno.

En los párrafos anteriores hemos realizado una breve introducción a los aspectos generales de la lógica borrosa útiles para la comprensión de los controladores basados en dicha lógica. A continuación haremos una introducción a la estructura de los sistemas de control borroso.

Los **sistemas de control borroso** utilizan las expresiones de lógica borrosa para formular reglas orientadas al control de sistemas. Es decir, los sistemas de control borroso combinan unas variables de entrada, que producen unas variables de salida por medio de un grupo de reglas (**base de reglas**).

La estructura del controlador borroso a construir es la siguiente:

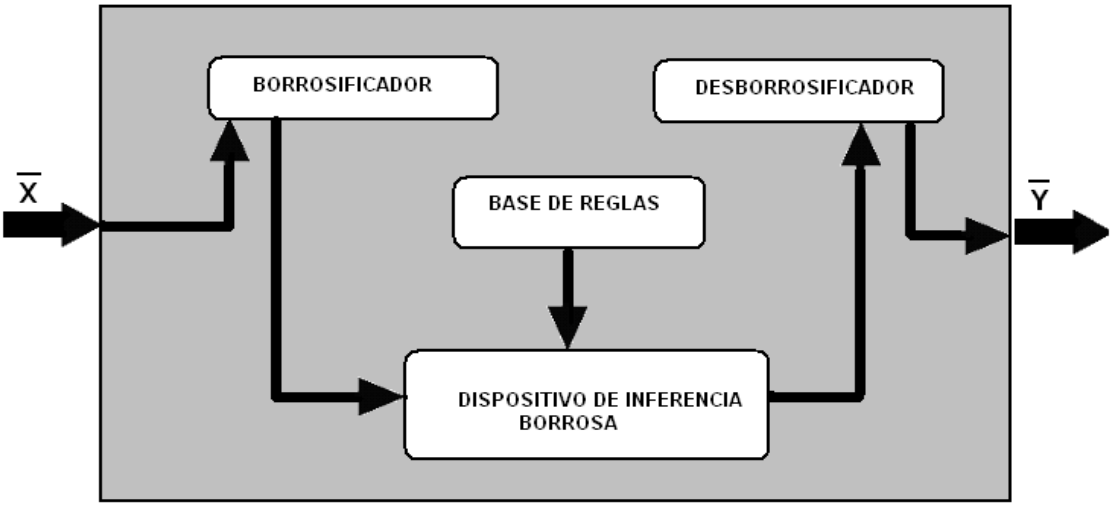


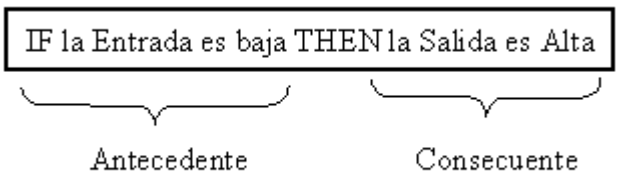
Fig. 10

Proseguiremos con el estudio de los elementos que se pueden encontrar en ella:

Base de reglas

La base de reglas es la manera que tiene el sistema difuso de guardar el conocimiento lingüístico que le permiten resolver el problema para el cual ha sido diseñado.

Está compuesta por un grupo de reglas en la cual cada regla es de **tipo IF (antecedente) THEN (consecuente)**. Las reglas combinan uno o más conjuntos borrosos de entrada, llamados antecedentes, y le asocian un conjunto de borroso de salida, llamado consecuente. Los conjuntos borrosos de la premisa se asocian mediante conjuntivas lógicas como y, o, etc.



El formato de reglas puede ser tanto de tipo **Mamdani** y de tipo **Sugeno**. En un sistema difuso tipo Mamdani tanto el antecedente como el consecuente de las reglas están dados por expresiones lingüísticas.

En cambio, en el tipo Sugeno la función de salida de la reglas es una función genérica de las variables de entrada.

- **Borrosificador**

La entrada de un sistema de lógica difusa normalmente es un valor numérico proveniente, por ejemplo, de un sensor. Para que este valor pueda ser procesado por el sistema difuso se hace necesario convertirlo a un "lenguaje" que el mecanismo de inferencia pueda procesar. Es decir, toma los valores provenientes del exterior y los convierte en valores "difusos". Estos valores difusos son los niveles de pertenencia de los valores de entrada a los diferentes conjuntos difusos en los cuales se ha dividido el universo de discurso de las diferentes variables de entrada al sistema.

Se pueden utilizar diferentes estrategias de borrosificación: borrosificador **singleton** y **no singleton**. El primero de ellos es el que más se utiliza.

- **Desborrosificador**

La salida que genera el dispositivo de inferencia es una salida difusa, lo cual significa que no puede ser interpretada por un elemento externo (por ejemplo un controlador) que sólo manipule información numérica. Para lograr que la salida del sistema difuso pueda ser interpretada por elementos que sólo procesen información numérica, hay que convertir la salida difusa mediante mecanismos de inferencia; este proceso lo realiza el desborrosificador.

Para ello existen diferentes métodos; si las reglas utilizadas son de tipo Mamdani podremos optar en el desborrosificador por máximo, desborrosificador por media de centros o desborrosificador por centro de área. En cambio, si las reglas son de tipo sugeno se utilizada la media ponderada de las salidas de cada reglas de la base de reglas.

- **Dispositivo de inferencia borrosa**

Teniendo los diferentes niveles de pertenencia arrojados por el borrosificador, los mismos deben ser procesados para generar una salida difusa. La tarea del sistema de inferencia es tomar los niveles de pertenencia y apoyado en la base de reglas generar la salida del sistema difuso.

2.3. Diseño del controlador

En esta sección se definirán los aspectos más importantes de los controladores borrosos a crear. De hecho, se diseñarán las características comunes de los tres controladores a crear, quedando excluidos tanto la base

de reglas como los parámetros de las funciones de inclusión de los conjuntos borrosos. A la hora de diseñar el controlador se hará uso de muchos de los conceptos introducidos en el capítulo anterior.

Los controladores serán implementados con la ayuda de la caja de herramientas “Fuzzy Logic Toolbox” de Matlab, mediante la cual se nos facilitará la tarea de implementación de los mismos, ya que se trata de una herramienta de fácil manejo para el usuario y eficiente al mismo tiempo.

Entradas y salidas del controlador borroso

Las **variables de entrada** del controlador borroso serán los valores devueltos por los sensores. Para ello, agruparemos los ocho sensores del robot Khepera II creando 4 grupos de sensores: izquierda, delante, derecha y atrás (*ver Fig.11*). De este modo se simplificará el controlador borroso a crear, reduciendo el número de entradas de ocho a cuatro.

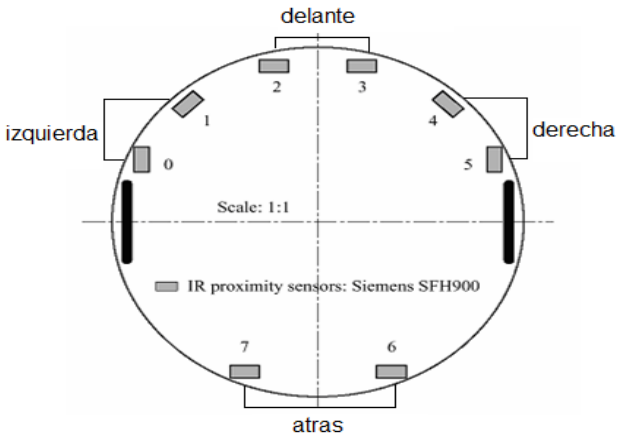


Fig. 11

Para cada variable de entrada, que estará definida en el rango de valores [0,1023], definiremos diversas particiones. En nuestro caso, el numero de particiones será de tres, es decir, tendremos tres subconjuntos borrosos, cada uno identificado por una etiqueta {cerca, medio, lejos}, correspondiente a la distancia en la que se encuentra la pared. Para ello, se deberá tener en cuenta que cuanto más alto es el valor devuelto por los sensores más cerca se encontrará la pared y viceversa. Cada uno de estos subconjuntos estará definido con una función de tipo triangular (*ver Fig.12*).

Por otro lado, las **variables de salida** de este controlador borroso serán la velocidad correspondiente a cada motor del robot. Es decir, tendremos dos salidas correspondientes a la velocidad del motor izquierdo y la velocidad del derecho, de forma que se controlará el movimiento del robot.

Para las variables de salida, las cuales toman valores en el rango [0 10], definiremos también tres particiones o subconjuntos borrosos, cada uno

identificado por una etiqueta {lento, medio, rápido}, referentes a la velocidad del motor correspondiente. Cada subconjunto, al igual que los de las entradas, estará definido con una función de tipo triangular (*ver Fig.12*).

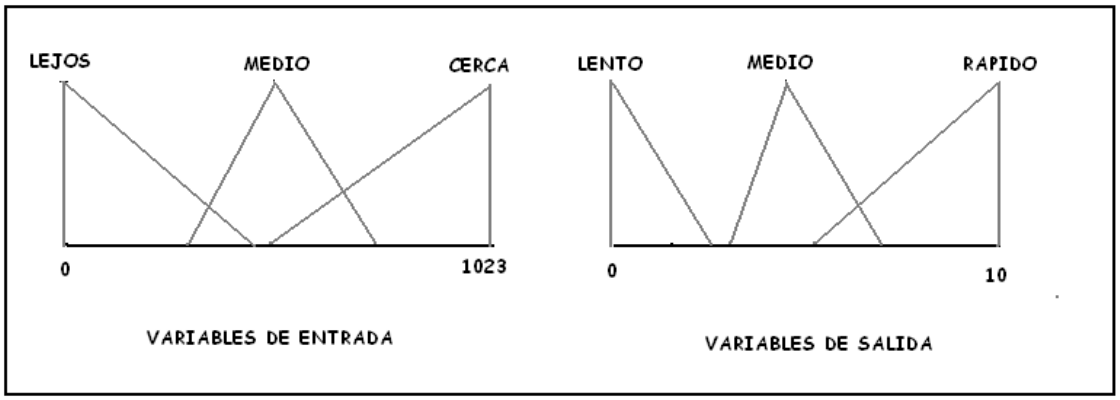


Fig. 12

Hay que tener en cuenta que la función de inclusión de cada conjunto borroso diferirá en base al comportamiento que se desea obtener.

Métodos de borrosificación, desborrosificación e inferencia

Hemos optado por el diseño de un sistema de lógica borrosa con *borrosificador* tipo singleton, *desborrosificador* por media de centros e *implicación borrosa* por la regla del mínimo.

Base de reglas

Seleccionamos el tipo de reglas a utilizar en el controlador, en principio podemos elegir entre las de tipo Mamdani o las de tipo Sugeno. Las de tipo Mamdani permiten expresar el conocimiento previo disponible sobre el sistema, expresando así el adquirido durante el proceso de optimización. Por su parte, las reglas de tipo Sugeno simplifican los cálculos de la salida, pero en general no resultan tan adecuadas para expresar el conocimiento de los expertos.

Como en este problema disponemos de un conocimiento intuitivo de las acciones de control a realizar sobre el sistema, nos decantaremos por el empleo de *reglas Mamdani*.

La base de reglas, las cuales asociarán a cada una de las posibles combinaciones de las entradas un valor de salida, se detallará en los apartados en que se describen los comportamientos, ya que variarán en torno a ellos.

2.4. Comportamientos básicos de navegación

2.4.1. Seguir pared

El objetivo en esta sección es que el Khepera II sea capaz de seguir una pared de manera efectiva. Es decir, realizando el seguimiento a una distancia apropiada de la pared y sin colisionar con ella (*ver Fig.13*).

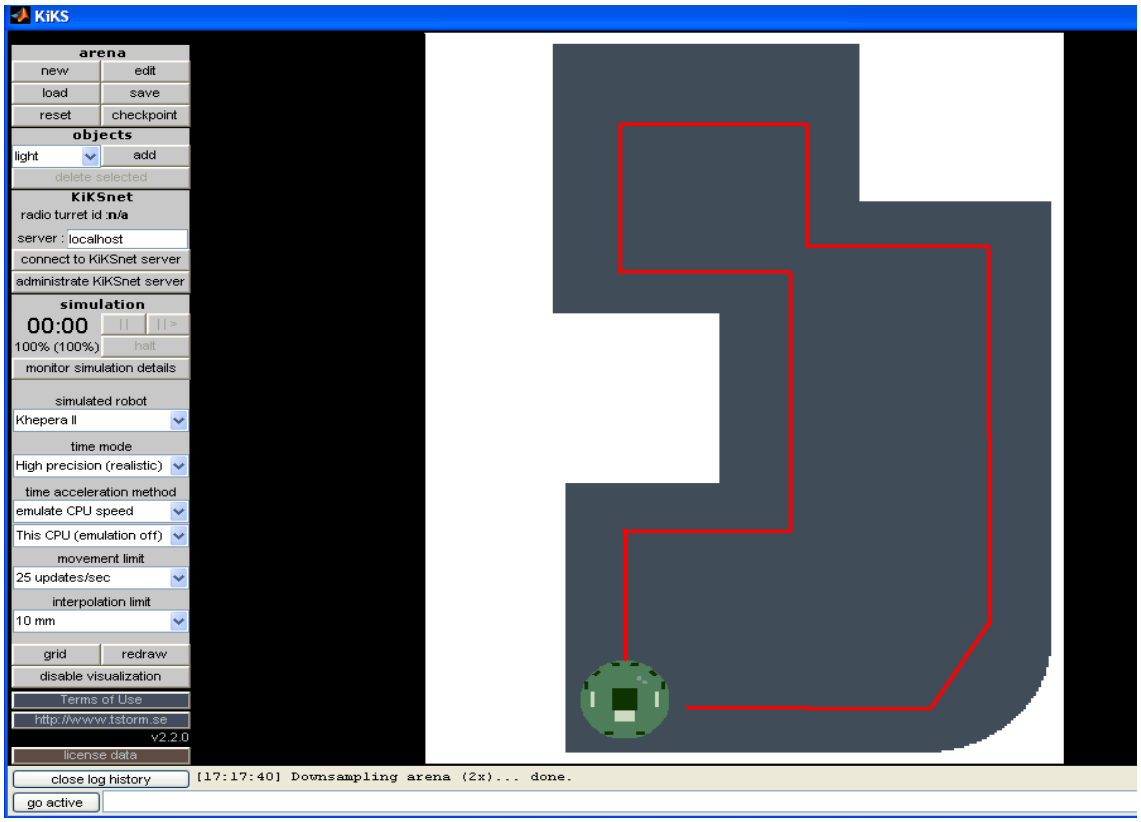


Fig. 13

El diseño que va a tener el controlador de seguimiento de pared se ha descrito en el apartado anterior, pero las reglas específicas del controlador se describirán en esta sección. Para la generación de las reglas se deberá tener en cuenta que en nuestro caso únicamente optaremos por implementar un controlador que sea capaz de realizar el seguimiento de una pared que se encuentre a la izquierda. Por esta razón, las reglas relevantes serán aquellas en las que intervengan los sensores izquierdos y delanteros.

La base de reglas contará con las siguientes reglas:

DEL \ IZQ	CERCA		MEDIO		LEJOS	
CERCA	MIZQ MDCHA	● ●	MIZQ MDCHA	● ●	MIZQ MDCHA	● ●
MEDIO	MIZQ MDCHA	● ●	MIZQ MDCHA	● ●	MIZQ MDCHA	● ●
LEJOS	MIZQ MDCHA	● ●	MIZQ MDCHA	● ●	MIZQ MDCHA	● ●

● LENTA ● MEDIA ● RÁPIDA ● NO HAY REGLA

Tabla 2

Con el diseño del controlador realizado se ha proseguido a probar dicho controlador. En la primera prueba, es decir, con los parámetros de las funciones con los valores iniciales por defecto, se ha podido ver que el controlador no actúa de una manera eficaz. Por ello, ha sido necesario editar los parámetros de las funciones de inclusión de las entradas izquierda y delante.

Los parámetros de cada función de inclusión de cada subconjunto borroso de entrada son los siguientes:

LEJOS	MEDIO	CERCA
[-409 0 200]	[200 400 600]	[400 1020 1420]

Tabla 3

Las funciones de inclusión de los conjuntos borrosos de las dos variables de salida son iguales, por consiguiente, los parámetros son:

LENTO	MEDIO	RÁPIDO
[-4 0 1.468]	[3.01 5.12 7.07]	[4.061 10.2 14.2]

Tabla 4

En la imagen siguiente se puede observar el recorrido seguido por el robot Khepera con el controlador creado.

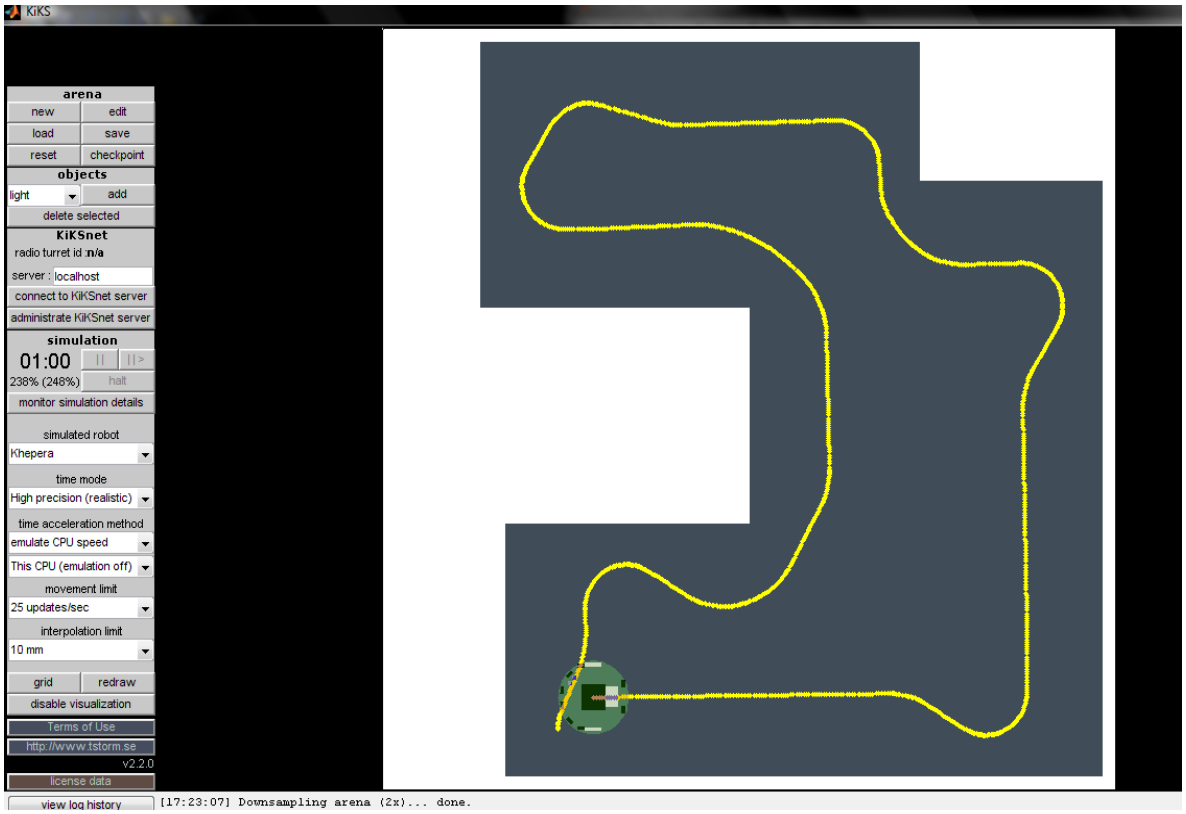


Fig. 14

2.4.2. Avanzar por pasillo

En esta fase la idea fundamental es hacer que el robot se mueva por un pasillo o corredor sin colisionar con los laterales ni atravesar puertas laterales. Además, si el robot llega a una esquina (*ver Fig.15*), la tiene que doblar sin colisionar, es decir, no tiene que verla como una puerta si no que sigue siendo el propio pasillo.

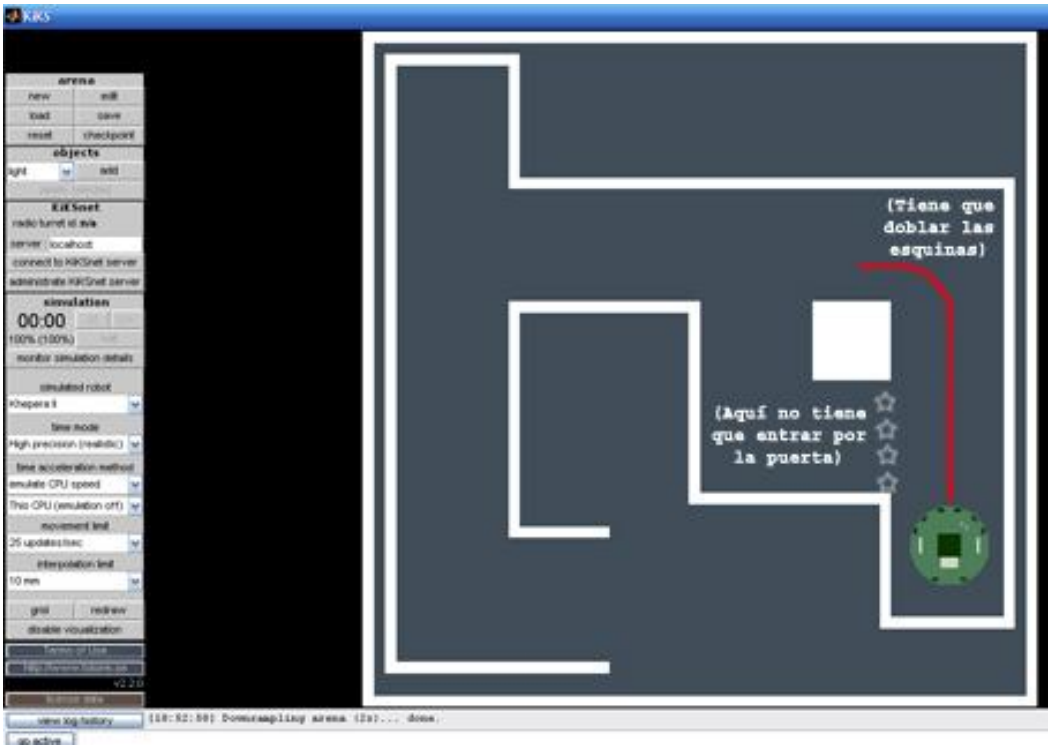


Fig. 15

En primer lugar, al igual que en el apartado anterior, pensaremos en las reglas para el controlador. Las reglas son las siguientes:

DCHA \ IZQ	CERCA		MEDIO		LEJOS	
CERCA	MIZQ MDCHA	● ●	MIZQ MDCHA	● ●	MIZQ MDCHA	● ●
MEDIO	MIZQ MDCHA	● ●	MIZQ MDCHA	● ●	MIZQ MDCHA	● ●
LEJOS	MIZQ MDCHA	● ●	MIZQ MDCHA	● ●	MIZQ MDCHA	● ●

● LENTA ● MEDIA ● RÁPIDA ● NO HAY REGLA

Tabla 5

IZQ								
DEL		CERCA		MEDIO		LEJOS		
	CERCA	MIZQ MDCHA	● ●	MIZQ MDCHA	● ●	MIZQ MDCHA	● ●	
	MEDIO	MIZQ MDCHA	● ●	MIZQ MDCHA	● ●	MIZQ MDCHA	● ●	
	LEJOS	MIZQ MDCHA	● ●	MIZQ MDCHA	● ●	MIZQ MDCHA	● ●	
● LENTA ● MEDIA ● RÁPIDA ● NO HAY REGLA								

Tabla 6

DCHA								
DEL		CERCA		MEDIO		LEJOS		
	CERCA	MIZQ MDCHA	● ●	MIZQ MDCHA	● ●	MIZQ MDCHA	● ●	
	MEDIO	MIZQ MDCHA	● ●	MIZQ MDCHA	● ●	MIZQ MDCHA	● ●	
	LEJOS	MIZQ MDCHA	● ●	MIZQ MDCHA	● ●	MIZQ MDCHA	● ●	
● LENTA ● MEDIA ● RÁPIDA ● NO HAY REGLA								

Tabla 7

La siguiente tabla (*ver Tabla 8*), a diferencia de las anteriores, refleja las reglas triples, es decir, las reglas en las que intervienen 3 entradas en lugar de 2.

IZQ	DEL	DCHA		
LEJOS	LEJOS	CERCA	MIZQ MDCHA	● ●
LEJOS	LEJOS	MEDIO	MIZQ MDCHA	● ●
● LENTA ● MEDIA ● RÁPIDA ● NO HAY REGLA				

Tabla 8

Las funciones de inclusión de la variable de entrada izquierda y la variable derecha son iguales. Los parámetros de cada función de inclusión de cada subconjunto borroso son los siguientes:

LEJOS	MEDIO	CERCA
[-409 0 150]	[150 600 989]	[990.5 1020 1430]

Tabla 9

Respecto a la variable delante, los parámetros de cada función de inclusión de cada subconjunto borroso son los siguientes:

LEJOS	MEDIO	CERCA
[-409 0 100]	[100 600 989]	[990.5 1020 1430]

Tabla 10

Las funciones de inclusión de los conjuntos borrosos de las dos variables de salida son iguales, los parámetros son:

LENTO	MEDIO	RÁPIDO
[-4 0 1]	[2 5.5 8]	[6 10 14]

Tabla 11

Mediante este diseño y probando con diferentes funciones de inclusión se ha alcanzado una solución aceptable, pero no óptima, ya que el robot roza la pared en algunas ocasiones. Luego será necesario optimizar este controlador mediante algoritmos genéticos. Estos serán explicados en un capítulo posterior.

2.4.3. Detectar puerta

En la siguiente sección se diseñará un controlador lógico con el objetivo de que el robot entre por una puerta. En este caso las reglas que se crearán difieren según el lado en la que se encuentra la puerta. Es decir, si la puerta por la que se requiere que el robot entre se encuentra a la izquierda, las reglas del controlador serán distintas a las del controlador para que el robot entre por la puerta derecha. Resumiendo, se necesitarán dos controladores.

En la siguiente figura se muestra el recorrido del robot cuando entra por una puerta izquierda. Al entrar por una puerta se deja atrás un pasillo, llegando a otro pasillo diferente.

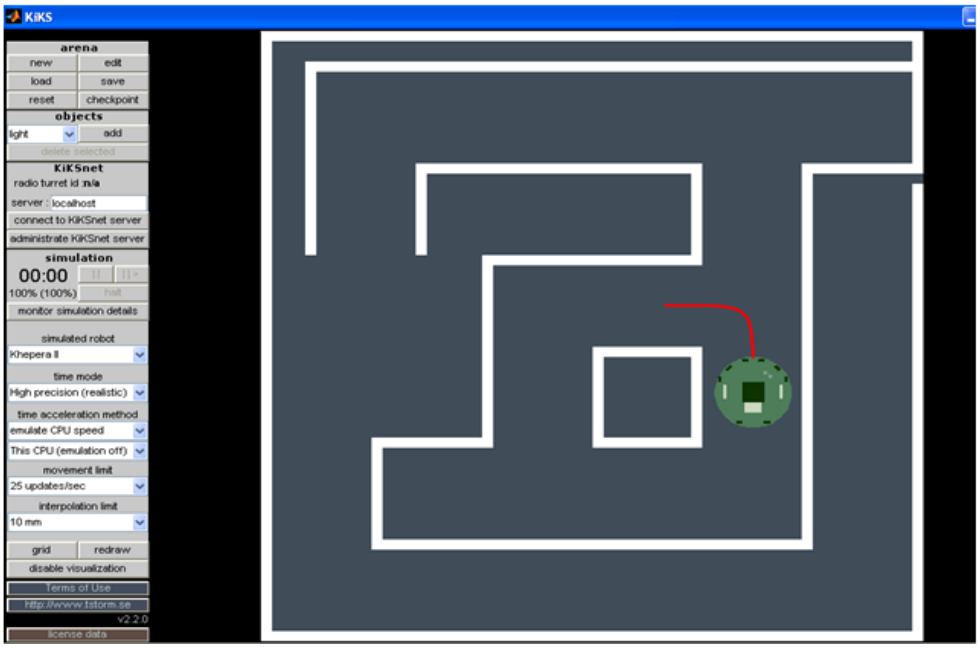


Fig. 16

Reglas para el controlador entrar por puerta izquierda:

IZQ	DEL							
		CERCA		MEDIO		LEJOS		
DEL	CERCA	MIZQ MDCHA	● ●	MIZQ MDCHA	● ●	MIZQ MDCHA	● ●	
	MEDIO	MIZQ MDCHA	● ●	MIZQ MDCHA	● ●	MIZQ MDCHA	● ●	
	LEJOS	MIZQ MDCHA	● ●	MIZQ MDCHA	● ●	MIZQ MDCHA	● ●	

● LENTA ● MEDIA ● RÁPIDA ● NO HAY REGLA

Tabla 12

Para realizar el controlador de una forma eficaz, aparte de las reglas dobles que hemos añadido, hemos optado por añadir un par de reglas triples.

IZQ	DEL	DCHA		
LEJOS	LEJOS	LEJOS	MIZQ MDCHA	● ●

● LENTA ● MEDIA ● RÁPIDA ● NO HAY REGLA

Tabla 13

Reglas para el controlador entrar por puerta derecha:

DEL \ DCHA	CERCA		MEDIO		LEJOS	
CERCA	MIZQ MDCHA	● ●	MIZQ MDCHA	● ●	MIZQ MDCHA	● ●
MEDIO	MIZQ MDCHA	● ●	MIZQ MDCHA	● ●	MIZQ MDCHA	● ●
LEJOS	MIZQ MDCHA	● ●	MIZQ MDCHA	● ●	MIZQ MDCHA	● ●

● LENTA ● MEDIA ● RÁPIDA ● NO HAY REGLA

Tabla 14

Las reglas triples en este caso son las siguientes:

IZQ	DEL	DCHA		
LEJOS	LEJOS	LEJOS	MIZQ MDCHA	● ●

● LENTA ● MEDIA ● RÁPIDA ● NO HAY REGLA

Tabla 15

Las funciones de inclusión de la variable de entrada izquierda y derecha son iguales, los parámetros de cada función de inclusión de cada subconjunto borroso son los siguientes:

LEJOS	MEDIO	CERCA
[-409 0 150]	[202 505 791]	[602.2 1020 1430]

Tabla 16

Las funciones de inclusión de los conjuntos borrosos de las dos variables de salida son iguales, y los parámetros son:

LENTO	MEDIO	RÁPIDO
[-4 0 1.971]	[1.96 5 7.99]	[5.992 10 14]

Tabla 17

Por otro lado, es necesario mencionar que este controlador únicamente es válido para poder entrar por una puerta, en el caso de que el robot se encuentre en medio de un pasillo sin ninguna puerta cerca, este controlador causaría que el robot se colisionase contra la pared.

2.5. Controlador para recorrido de pasillo con puertas

En las etapas anteriores, se crearon varios comportamientos diferenciados para el robot, así pues, esta etapa consistirá en acoplar de alguna manera esos comportamientos en un único controlador capaz de realizar un recorrido determinado (*ver fig.17*).

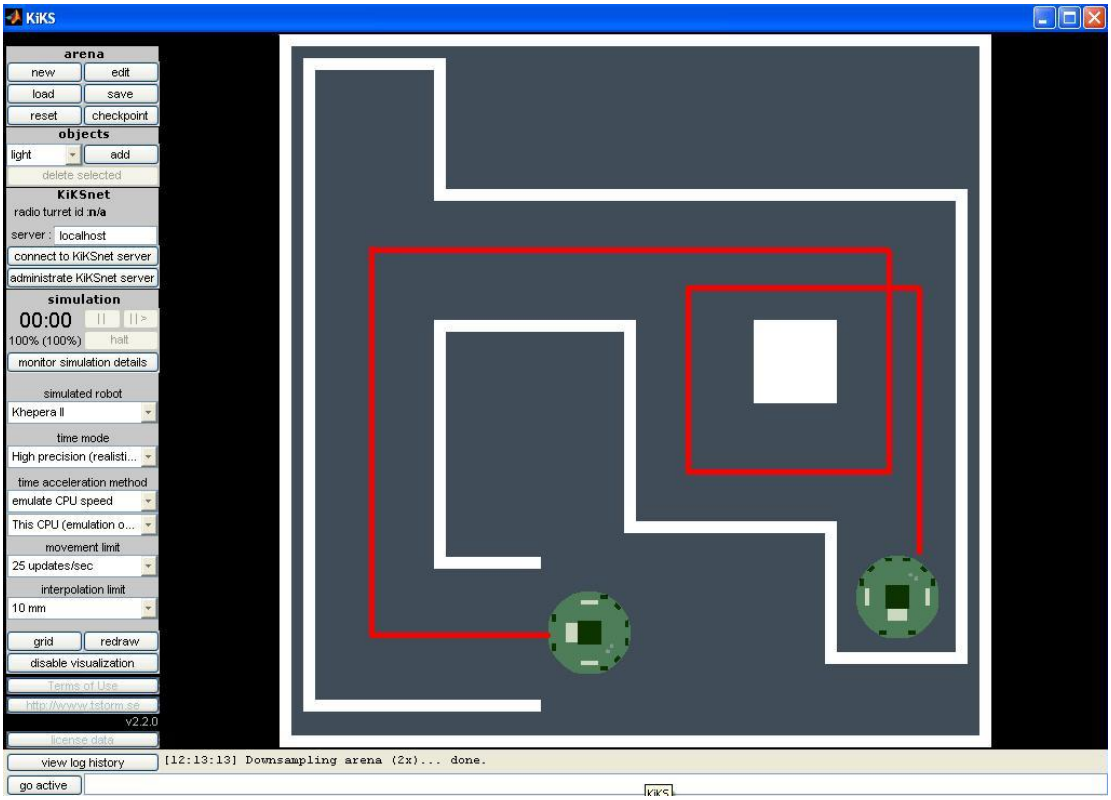


Fig. 17

Una de las principales dificultades a la hora de diseñar este controlador es la necesidad de saber la posición del robot en el tablero. Es decir, las coordenadas X e Y del laberinto, puesto que necesitamos saber en qué rango de posiciones debe actuar el controlador “puerta” o “pasillo”. Puesto que en nuestro caso no es posible obtener la posición absoluta del robot, utilizaremos la **odometría** para obtener la posición relativa del robot.

La **odometría** es usada por los robots móviles con ruedas para estimar su posición relativa a su localización inicial durante la navegación. La odometría se basa en ecuaciones simples que se pueden implementar fácilmente y que utilizan datos de encoders situados en las ruedas del robot. Sin embargo, la idea fundamental de la odometría es la integración de información incremental del movimiento a lo largo del tiempo, lo cual conlleva una inevitable

acumulación de errores, tanto sistemáticos (desalineación de las ruedas, desigual diámetro de las ruedas...) como no sistemáticos (encuentro con objetos no deseados, derrape en las ruedas: aceleración, giros rápidos, etc.).

Así pues, programamos dos funciones diferentes que efectúan una serie de cálculos dándonos como resultado final la posición X e Y del robot. Una de las funciones tomará en cuenta el tiempo que ha tardado el robot en cambiar en posición, mientras que la otra no.

En primer lugar, realizamos un **test** para comprobar cuál de las 2 funciones de posición nos brinda mejores resultados. Tras realizar 10 pruebas con cada una de ellas, consistentes en avanzar por un pasillo hasta detectar la pared frontal, obtuvimos los siguientes valores:

1ª: TENIENDO EN CUENTA EL TIEMPO		2ª: SIN TENER EN CUENTA EL TIEMPO	
X	**Y**	**X**	**Y**
327.43	159.96	293.72	132.27
324.99	176.39	301.57	119.02
318.90	156.02	294.22	129.29
354.39	165.68	295.64	118.18
308.37	188.89	290.55	139.73
333.62	164.02	296.25	127.94
328.51	136.96	302.54	114.51
320.68	175.13	297.16	131.85
370.94	155.35	290.54	135.04
365.90	142.69	291.68	135.41

Tabla 18

Como se puede apreciar en las tablas anteriores, los valores difieren menos utilizando el 2º algoritmo, es decir, sin tener en cuenta el tiempo, luego se puede pensar que la primera función esta mejor programada para obtener la posición relativa, por consiguiente ésta será la elegida para obtener los rangos en los que se dará el “cambio de comportamiento”.

Una vez definida la función para obtener la posición del robot, procederemos a identificar los **rangos de posiciones** de nuestro recorrido en las que se dará el “cambio de comportamiento” o se realizará la “toma de decisión”. Es decir, obtendremos aquellos rangos en los que se decidirá si el robot debe de entrar por la puerta o seguir por el pasillo. En la imagen que se muestra a continuación se especifican estas posiciones.

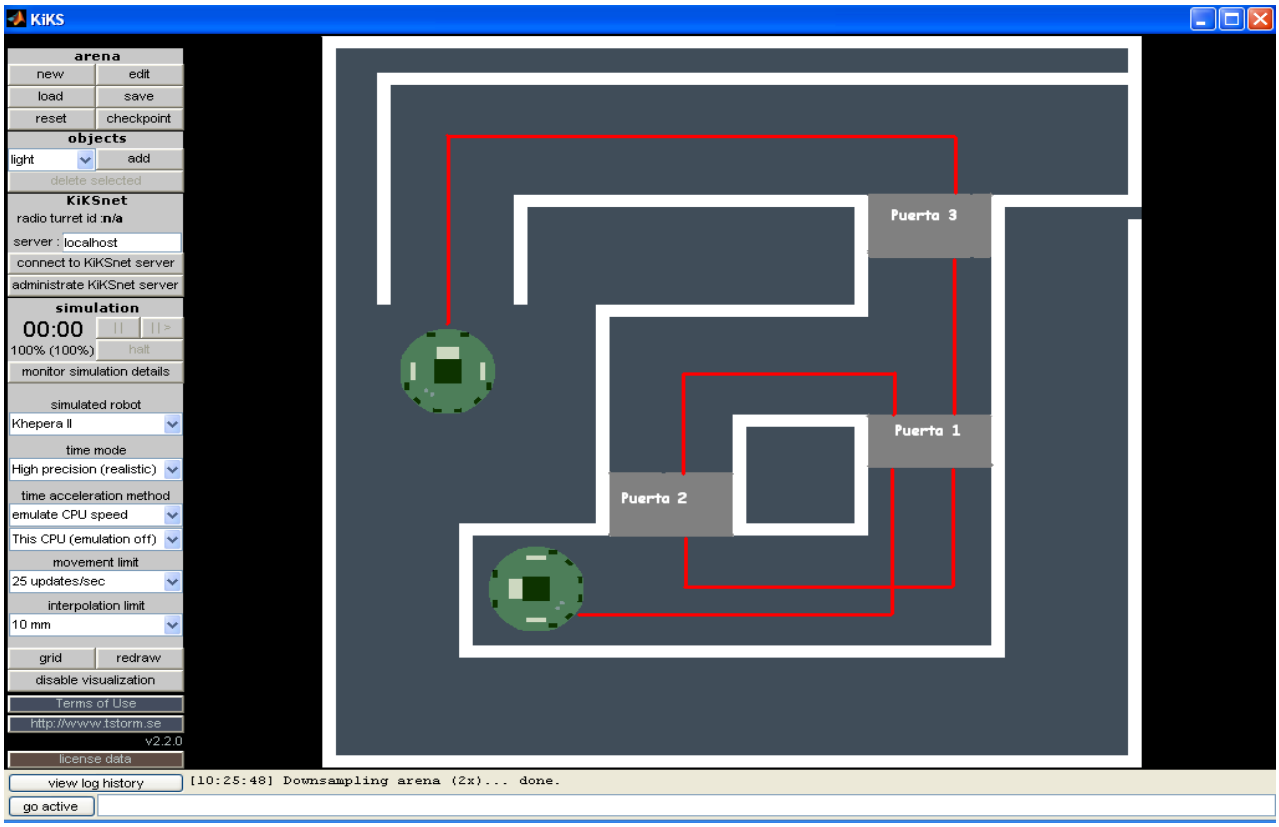


Fig. 18

Así pues, obtenemos el rango para la PUERTA 1:

Rango_Puerta1	
X	Y
290-303	114-140

Tabla 19

Si observamos el recorrido que se quiere realizar se puede ver que el robot pasará dos veces por este lugar. La primera vez que pase por el rango de posiciones de Puerta 1, se le debe informar al robot que entre por la puerta, en cambio, la segunda vez que el Khepera II se encuentre dentro del rango seguirá el pasillo. Para poder lograr este comportamiento haremos uso de la distancia recorrida. La distancia recorrida nos será útil para poder observar en que parte del recorrido está, es decir, si ya ha entrado una vez por la puerta 1, o en cambio, si es la todavía no ha entrado. De esta forma si la distancia recorrida es mayor que una dada, el robot seguirá por el pasillo ignorando la puerta, en el otro caso entrará por ella. Estableceremos el umbral de la distancia en 1030.

A continuación, procederemos a obtener el rango de posiciones de PUERTA 2 y PUERTA 3. Cuando el robot se encuentre dentro de estos rangos,

la decisión que deberá adoptar el robot será la de entrar por la siguiente puerta. De esta forma, se conseguirá que el robot efectúe el recorrido deseado.

Rango_Puerta2	
X	**Y**
91-112	51-76

Tabla 20

Rango_Puerta3	
X	**Y**
234-277	280-329

Tabla 21

Una vez que se han obtenido los rangos en los que el robot deberá entrar por una puerta, nos encontramos con el problema siguiente: si el robot entra por una puerta deja el pasillo en el que estaba y entra a otro pasillo. Luego, una vez que se ha entrado por la puerta, el robot deberá volver a seguir un pasillo. Luego debemos realizar de nuevo un cambio de controlador, deberá volver a actuar el controlador de pasillo. Para ello primeramente se deben identificar los rangos de posiciones para el cambio de controlador (ver fig.19).

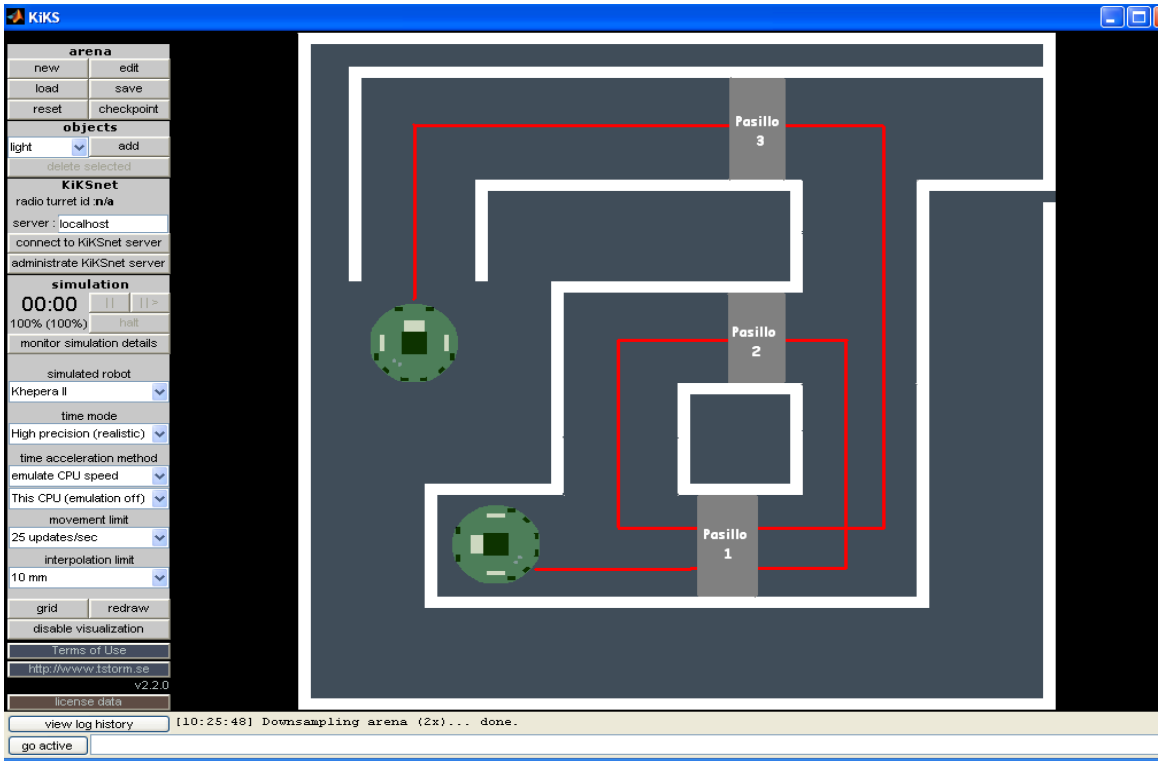


Fig. 19

En primer lugar, se obtendrán las posiciones del rango Pasillo 2. Este cambio de controlador se dará cuando el robot entre por la primera puerta y permitirá que el robot avance por el segundo pasillo.

Rango_Pasillo2	
X	**Y**
185-222	183-206

Tabla 22

A continuación, se obtendrán las posiciones del rango Pasillo 1. Este cambio de controlador se dará cuando el robot entre por la segunda puerta y permitirá que el robot avance por el pasillo del principio.

Rango_Pasillo1	
X	**Y**
156-207	-30 -20

Tabla 23

Finalmente, se obtendrán las posiciones del rango Pasillo 3. Este cambio de controlador se dará cuando el robot entre por la tercera puerta y permitirá que el robot avance por el pasillo 3.

Rango_Pasillo3	
X	**Y**
219-249	360-382

Tabla 24

Para finalizar con los rangos de posiciones, obtendremos un último rango en el que se especificará si el robot ha llegado a la posición final dando por finalizado su recorrido. El rango de posiciones de la meta o posición final del robot serán las siguientes.

Rango_Final	
X	**Y**
-71 - -42	140-181

Tabla 25

Una vez especificados todos los rangos de “cambio de controlador”, será necesario definir un umbral para que una vez activado el comportamiento entrar por puerta, el robot empiece a entrar cuando esté seguro de que se trata de una puerta. La razón de esto se encuentra en que se puede dar el caso en

el que el robot puede ser informado de que debe de entrar por puerta antes de que se encuentre en ella, ya que como hemos mencionado anteriormente el modelo de odometría usado es algo inexacto. En este caso, la reacción del robot sería colisionar con la pared. Resumiendo, una vez que se le manda al robot entrar por la siguiente puerta, se activará el comportamiento entrar puerta cuando exista la certeza de que efectivamente hay una puerta.

Para obtener dicho umbral, se han realizado diferentes medidas de los sensores izquierdos existiendo una puerta en dicho lado. La siguiente gráfica muestra el resultado de las mediciones mencionadas:

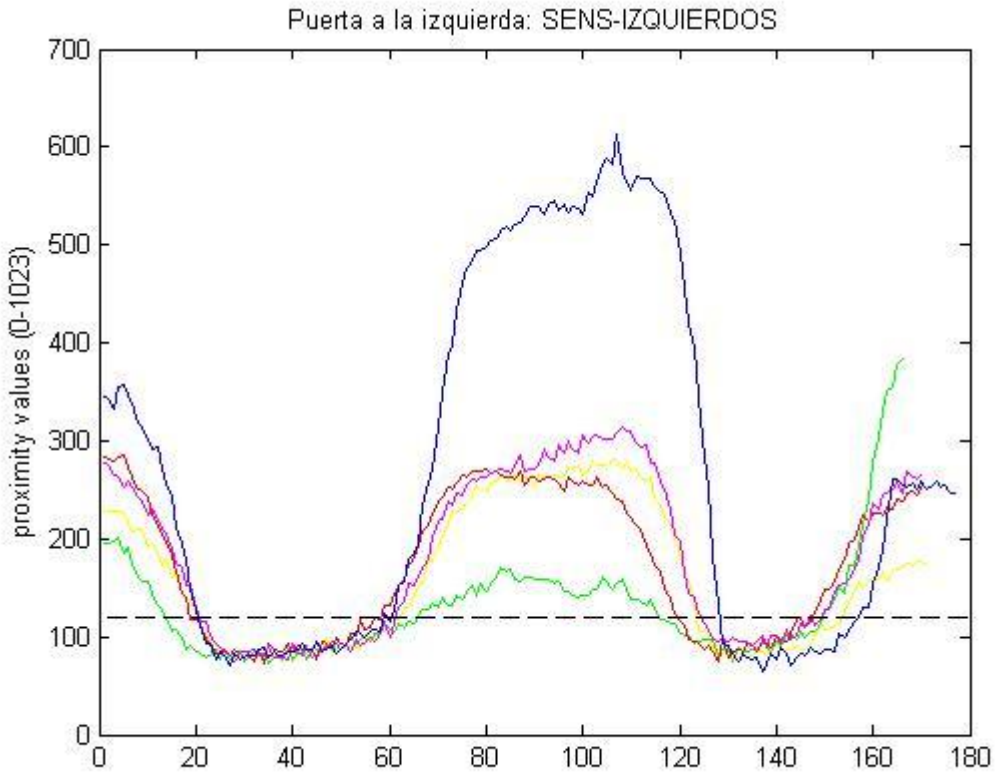


Fig. 20

Por lo tanto, como se puede observar en la gráfica, en el caso de que el valor de los sensores izquierdos sea menor que 120, podemos estar seguros de que hay una puerta, y el robot entraría por puerta de darse el caso. Este umbral es válido análogamente para el lado derecho.

2.6. Conclusión del capítulo

En este capítulo se han diseñado diferentes controladores difusos con los cuales hemos sido capaces de implementar de manera aceptable el recorrido de pasillo con puertas. No obstante, el robot no realiza el recorrido de una forma óptima, ya que el robot roza la pared en ciertas ocasiones y no realiza el recorrido en el menor tiempo posible. Por esta razón, resulta imprescindible

que tomando los controladores puerta y pasillo diseñados en este capítulo realicemos la optimización de dichos controladores mediante algoritmos genéticos. La optimización ha sido implementada en el siguiente capítulo.

Capítulo 3. Optimización del controlador difuso mediante Algoritmos Genéticos

3.1. Introducción a los GA's

Los Algoritmos Genéticos (AG's) son métodos adaptativos que pueden usarse para resolver problemas de búsqueda y optimización. Están basados en el proceso genético de los organismos vivos. A lo largo de las generaciones, las poblaciones evolucionan en la naturaleza de acorde con los principios de la selección natural y la supervivencia de los más fuertes, postulados por Darwin (1859). Por imitación de este proceso, los Algoritmos Genéticos son capaces de ir creando soluciones para problemas del mundo real. La evolución de dichas soluciones hacia valores óptimos del problema depende en buena medida de una adecuada codificación de las mismas.

Un algoritmo genético consiste en una función matemática o una rutina de software que toma como entradas a los ejemplares y retorna como salidas cuáles de ellos deben generar descendencia para la nueva generación.

En la naturaleza los individuos de una población compiten entre sí en la búsqueda de recursos tales como comida, agua y refugio. Incluso los miembros de una misma especie compiten a menudo en la búsqueda de un compañero. Aquellos individuos que tienen más éxito en sobrevivir y en atraer compañeros tienen mayor probabilidad de generar un gran número de descendientes. Por el contrario individuos poco dotados producirán un menor número de descendientes. Esto significa que los genes de los individuos mejor adaptados se propagarán en sucesivas generaciones hacia un número de individuos creciente. La combinación de buenas características provenientes de diferentes ancestros, puede a veces producir descendientes "súper individuos", cuya adaptación es mucho mayor que la de cualquiera de sus ancestros. De esta manera, las especies evolucionan logrando unas características cada vez mejor adaptadas al entorno en el que viven.

Los Algoritmos Genéticos usan una analogía directa con el comportamiento natural. Trabajan con una población de individuos, cada uno de los cuales representa una solución factible a un problema dado. A cada individuo se le asigna un valor ó puntuación, relacionado con la bondad de dicha solución. En la naturaleza esto equivaldrá al grado de efectividad de un organismo para competir por unos determinados recursos. Cuanto mayor sea la adaptación de un individuo al problema, mayor será la probabilidad de que el mismo sea seleccionado para reproducirse, cruzando su material genético con otro individuo seleccionado de igual forma. Este cruce producirá nuevos individuos - descendientes de los anteriores - los cuales comparten algunas de las características de sus padres. Cuanto menor sea la adaptación de un individuo, menor será la probabilidad de que dicho individuo sea seleccionado para la reproducción, y por tanto de que su material genético se propague en sucesivas generaciones.

De esta manera se produce una nueva población de posibles soluciones, la cual reemplaza a la anterior y verifica la interesante propiedad de que contiene una mayor proporción de buenas características en comparación con la población anterior. Así a lo largo de las generaciones las buenas características se propagan a través de la población. Favoreciendo el cruce de los individuos mejor adaptados, van siendo exploradas las áreas más prometedoras del espacio de búsqueda. Si el Algoritmo Genético ha sido bien diseñado, la población convergerá hacia una solución óptima del problema.

El poder de los Algoritmos Genéticos proviene del hecho de que se trata de una técnica robusta, y pueden tratar con éxito una gran variedad de problemas provenientes de diferentes áreas, incluyendo aquellos en los que otros métodos encuentran dificultades. Si bien no se garantiza que el Algoritmo Genético encuentre la solución óptima del problema, existe evidencia empírica de que se encuentran soluciones de un nivel aceptable, en un tiempo competitivo con el resto de algoritmos de optimización combinatoria. En el caso de que existan técnicas especializadas para resolver un determinado problema, lo más probable es que superen al Algoritmo Genético, tanto en rapidez como en eficacia. El gran campo de aplicación de los Algoritmos Genéticos se relaciona con aquellos problemas para los cuales no existen técnicas especializadas. Incluso en el caso en que dichas técnicas existan, y funcionen bien, pueden efectuarse mejoras de las mismas hibridizándolas con los Algoritmos Genéticos.

3.2. Pseudo-código del algoritmo genético simple

```
BEGIN /* Algoritmo Genético Simple */
        ● Generar una población inicial.
        ● Computar la función de evaluación de cada individuo.
        WHILE NOT Terminado DO
                BEGIN /* Producir nueva generación */
                FOR tamaño población/2 DO
                        BEGIN /*Ciclo Reproductivo */
                        ● Seleccionar dos individuos de la anterior generación, para el cruce
                        (probabilidad de selección proporcional a la función de evaluación del
                        individuo).
                        ● Cruzar con cierta probabilidad los dos individuos obteniendo dos
                        descendientes.
                        ● Mutar los dos descendientes con cierta probabilidad.
                        ● Computar la función de evaluación de los dos descendientes mutados.
                        ● Insertar los dos descendientes mutados en la nueva generación.
                        END
                IF la población ha convergido THEN
                        ● Terminado = TRUE
                END
END
```

Como se verá a continuación, se necesita una codificación o representación del problema, que resulte adecuada al mismo. Además se requiere una función de ajuste ó adaptación al problema, la cual asigna un

número real a cada posible solución codificada. Durante la ejecución del algoritmo, los padres deben ser seleccionados para la reproducción, a continuación dichos padres seleccionados se cruzarán generando dos hijos, sobre cada uno de los cuales actuará un operador de mutación. El resultado de la combinación de las anteriores funciones será un conjunto de individuos (posibles soluciones al problema), los cuales en la evolución del Algoritmo Genético formarán parte de la siguiente población.

3.3. Pasos para construir nuestro algoritmo genético

3.3.1. Diseñar una representación/codificación

Se supone que los individuos (posibles soluciones del problema), pueden representarse como un conjunto de parámetros (que denominaremos genes), los cuales agrupados forman una ristra de valores (a menudo referida como cromosoma). Si bien el alfabeto utilizado para representar los individuos no debe necesariamente estar constituido por el {0, 1}, buena parte de la teoría en la que se fundamentan los Algoritmos Genéticos utiliza dicho alfabeto.

En nuestro problema necesitamos obtener como solución los valores de los vértices de cada uno de los triángulos de entrada/salida del archivo .fis del controlador.

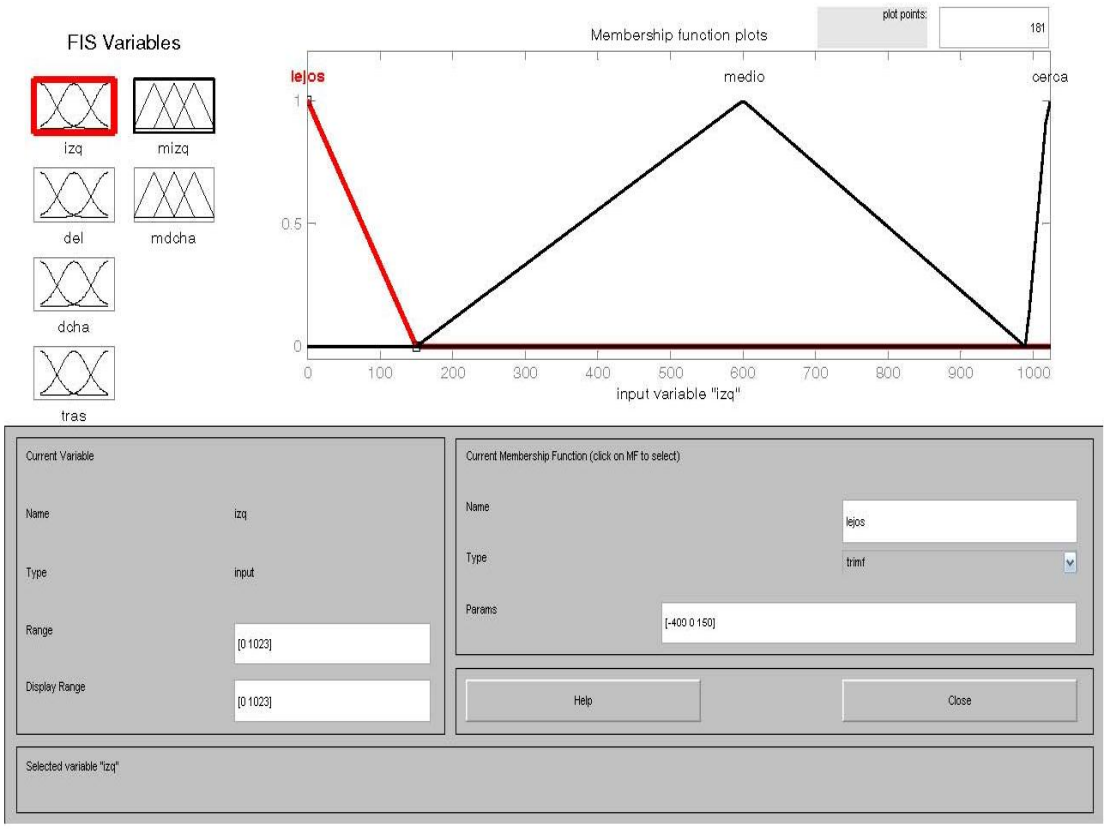


Fig. 21

Por lo tanto, como tenemos 3 variables de entrada (una por cada uno de los grupos de sensores {izq, del, dcha}) que a su vez tienen 3 etiquetas cada una ({lejos, medio, cerca}) y 2 variables de salida (las velocidades {mizq, mdcha}) las cuales tienen otras 3 etiquetas ({lenta, media, rápida}), son necesarios:

(3 variables de entrada *3 etiquetas *3 vértices) + (2 variables de salida *3 etiquetas * 3 vértices) = 45

⇒ Así pues, nuestro cromosoma estará compuesto por **45 genes**.

Además de tener 45 genes, otra de las cuestiones a la hora de diseñar el algoritmo genético es el número de bits que ocupará el cromosoma, y por lo tanto, cada uno de los genes. En nuestro problema, los rangos de valores que pueden tomar las variables de entrada y las de salida son diferentes, por lo que no tienen por qué representarse con el mismo número de bits. Dicho ésto, escogemos el menor número de bits que alcance a representar el rango correspondiente:

Variable de entrada: 0~1023 ⇒ 1023<210 ⇒ **10 bits**
Variable de salida: 0~10 ⇒ 10<24 ⇒ **4 bits**

⇒ En total, nuestro cromosoma estará compuesto por 27 genes de 10 bits y 18 genes de 4 bits.

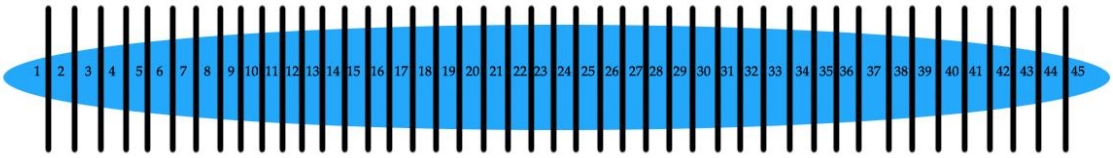


Fig. 22

En términos biológicos, el conjunto de parámetros representando un cromosoma particular se denomina fenotipo. El fenotipo contiene la información requerida para construir un organismo, el cual se refiere como genotipo. Los mismos términos se utilizan en el campo de los Algoritmos Genéticos. La adaptación al problema de un individuo depende de la evaluación del genotipo. Esta última puede inferirse a partir del fenotipo, es decir puede ser computada a partir del cromosoma, usando la función de evaluación.

3.3.2. Decidir cómo inicializar una población

A la hora de tomar esta decisión, pensamos en que teníamos ya varios controladores (3) implementados de las pruebas realizadas en fases anteriores que no funcionaban bien, obviamente, pero se asemejaban a lo que debía

hacer el robot. Si en el momento de inicializar la población incorporamos estos controladores como individuos, lo más probable es que los valores sean mejores que si generamos toda la población inicial aleatoriamente, y por lo tanto, que el algoritmo genético aumente así su probabilidad de "converger" (llegar a la solución esperada).

3.3.3. Diseñar una forma de evaluar un individuo

Este es sin duda el apartado más importante y costoso en la fase de diseño del algoritmo genético, ya que si la función de evaluación no está bien planteada conforme a las necesidades de nuestro problema no llegaremos a alcanzar la solución del mismo. Dado un cromosoma particular, la función de evaluación le asigna un número real, que se supone refleja el nivel de adaptación al problema del individuo representado por dicho cromosoma.

En nuestro caso, esta función consistirá en simular el recorrido del robot utilizando el archivo .fis con los parámetros del cromosoma y devolver el valor de adaptación del individuo representado por dicho cromosoma. Los factores en los que hemos pensado para calcular este valor de adaptación son los siguientes:

- La existencia de colisión, es decir, si el robot se choca contra la pared y no continúa con el recorrido.
- El número de veces que el robot roza con alguna de las paredes.
- El número de puntos de control (checkpoints) que el robot alcanza.
- La distancia recorrida por el robot (recogida de los encoders).
- El tiempo invertido en el recorrido (en caso de que no llegue al final del recorrido será asignado un tiempo máximo por defecto).

La fórmula quedaría de la siguiente manera, teniendo en cuenta que los valores de los diferentes factores serán mostrados en el apartado de resultados (*5.4. Resultados de las simulaciones*).

Fitness = (colision*f_colision+roces*f_roces + num_check*f_check + distancia*f_distancia – tiempo *f_tiempo);

3.3.4. Diseñar un operador de mutación adecuado

El operador de mutación se aplica a cada hijo de manera individual, y consiste en una alteración aleatoria. Para nuestro problema, el operador de mutación escogido se basa en una operación simple que consiste en aumentar

o disminuir el valor del gen en un rango adecuado al valor que puede tomar dicho gen. Es decir, para las entradas, las cuales varían en el intervalo de valores 0~1023 el **factor del rango (F)** de mutación será 50, y para las salidas, en el intervalo 0~10, será 1.

Así pues, este operador consistirá en generar un valor aleatorio en el rango [-F, F] y sumárselo al valor del gen a mutar.

Ejemplo de aplicación del operador de mutación

Sea el cromosoma siguiente, "C", el individuo en tratamiento, y el número 17 el número de gen a mutar obtenido aleatoriamente:

12	378	645	333	400	899	291	489	977	28	245	654	39	876	932
35	100	250	78	90	600	400	788	960	13	129	333	1	2	3
2	4	6	3	6	7	2	5	6	3	6	7	6	8	9

Fig. 23

Tras aplicar el operador de mutación al gen correspondiente, el cromosoma "C" quedaría:

x = aleatorio {-50,+50} = -3

12	378	645	333	400	899	291	489	977	28	245	654	39	876	932
35	97	250	78	90	600	400	788	960	13	129	333	1	2	3
2	4	6	3	6	7	2	5	6	3	6	7	6	8	9

Fig. 24

3.3.5. Diseñar un operador de cruce adecuado

Otro de los puntos clave a la hora de diseñar bien el algoritmo genético, es implementar un buen operador de cruce acorde a las necesidades de nuestro problema.

Generalmente, en muchos de los problemas resueltos mediante este tipo de técnicas, se suele utilizar el operador básico de **cruce monopunto**, consistente en coger dos padres seleccionados y cortar sus ristras de cromosomas en una posición escogida al azar, para producir dos subristras iniciales y dos subristras finales. Después se intercambian las subristras finales, produciéndose dos nuevos cromosomas completos. Ambos descendientes heredan genes de cada uno de los padres.

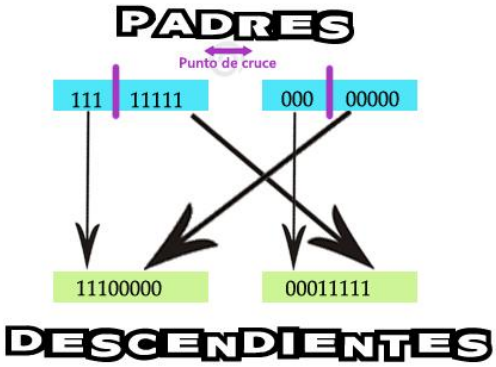


Fig. 25

Otro tipo de operador de cruce es el **cruce basado en 2 puntos**, el cual es parecido al anterior salvo que en vez de cruzar los cromosomas por un único punto de cruce, sólo se cruzan los genes comprendidos entre el primer punto y el segundo. Esto se puede ver claramente en la siguiente imagen.

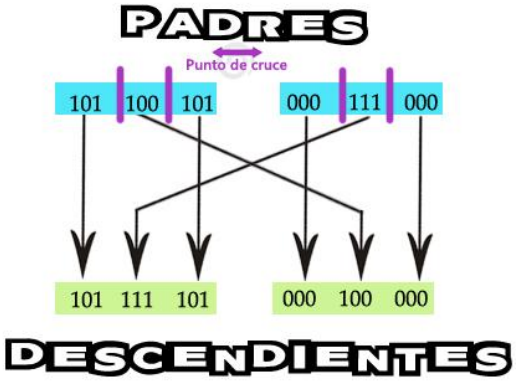


Fig. 26

Hay muchos tipos diferentes para este operador, desde los más básicos, comentados anteriormente, hasta otros más complejos o modificaciones de los básicos. El nuestro será un operador de cruce aritmético **para representación real BLX-α,** este consiste en:

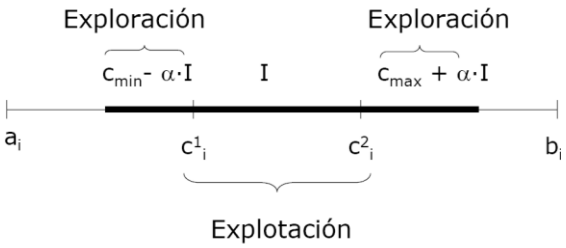
Dados 2 cromosomas:
C1= (c_{11},…, c_{1n}) y **C2**= (c_{21},…, c_{2n}),

BLX-αgenera dos descendientes:

Hk= (h_{k1},…, h_{ki},…, h_{kn}), k =1,2, donde cada gen h_{ki} se genera aleatoriamente en el intervalo: [C_{min}-I·α, C_{max}+ I·α]

C_{max}= max {c_{1i}, c_{2i}}
C_{min}= min {c_{1i}, c_{2i}}

I = C_{max}-C_{min}, α∈[0,1]



En un principio, el valor propuesto para el factor **α sería de 0,3**.

3.3.6. Decidir cómo seleccionar los individuos para ser padres

Durante la fase reproductiva se seleccionan los individuos de la población para cruzarse y producir descendientes, que constituirán, una vez mutados, la siguiente generación de individuos. La selección de padres se efectúa al azar usando un procedimiento que favorezca a los individuos mejor adaptados, ya que a cada individuo se le asigna una probabilidad de ser seleccionado que es proporcional a su función de adaptación. Este procedimiento se dice que está basado en la ruleta sesgada.

Según dicho esquema, los individuos bien adaptados se escogerán probablemente varias veces por generación, mientras que los pobremente adaptados al problema, no se escogerán más que de vez en cuando.

Una vez seleccionados dos padres, sus cromosomas se combinan, utilizando habitualmente los operadores de cruce y mutación anteriormente explicados.

3.3.7. Decidir cómo reemplazar a los individuos

La política de reemplazo escogida es **elitista**. En ciertas ocasiones puede suceder que tras el cruce y la mutación, perdamos el cromosoma con mejor adaptación. Este método copia el mejor cromosoma o alguno de los mejores directamente en la nueva población, en nuestro caso los **5 cromosomas** mejor adaptados. El resto se realiza de la misma forma que hemos visto anteriormente, cruzando los cromosomas de la antigua generación. El elitismo puede mejorar el funcionamiento de los algoritmos genéticos al evitar que se pierda la mejor solución. Una variación del elitismo es que el mejor cromosoma solo se copie a la siguiente generación en caso de que tras una reproducción/mutación no se haya generado un cromosoma mejor.

Capítulo 4. Resultados experimentales

4.1. Introducción a los simuladores

4.1.1. Simulador Kiks

A lo largo del proyecto se han detectado incongruencias entre el simulador KIKS y el robot real. De esta manera, comportamientos que en el robot real se ejecutan de una manera aceptable fallan al ejecutarlos en el simulador KIKS.

Tomando como punto de partida en controlador pasillo implementado en el capítulo anterior, el cual ha sido probado con el robot real con resultados aceptables, se ha proseguido a probarlo en dicho simulador. El resultado de esta simulación es una colisión contra la pared (*ver fig.27*), quedando explicito que el controlador falla en el simulador KIKS.

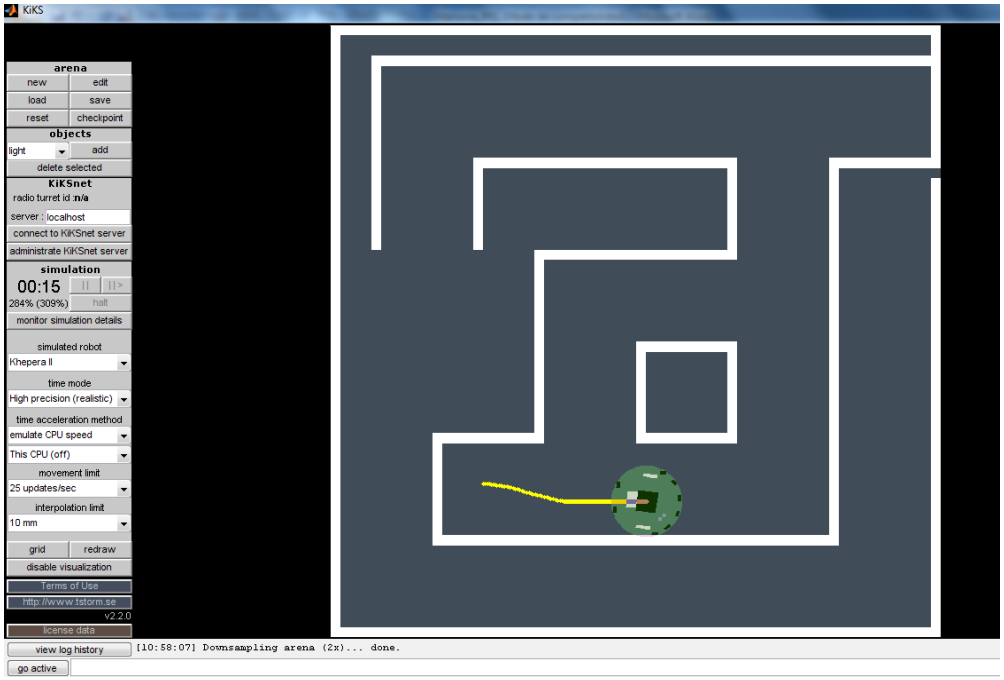


Fig. 27

El motivo del fallo del controlador se encuentra en la diferencia de las medidas sensoriales de los dos robots (*ver fig.28 y fig.29*). Por ello, para que el controlador funcionase en el simulador KIKS sería necesario adaptar los sensores. Sin embargo, como se observa en las gráficas la obtención de los sensores del simulador tiene muchos picos, al contrario de la linealidad con la que se obtienen del robot real. Por el contrario, analizando la gráfica de los sensores delanteros se observa la enorme diferencia existente entre las dos mediciones. Por estas razones es complicado adaptar los sensores del simulador a los del robot.

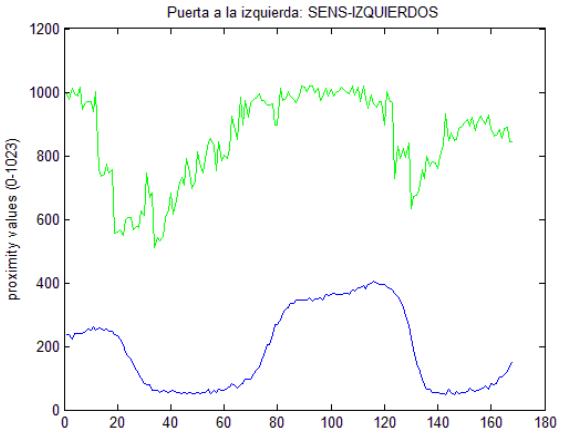


Fig. 28

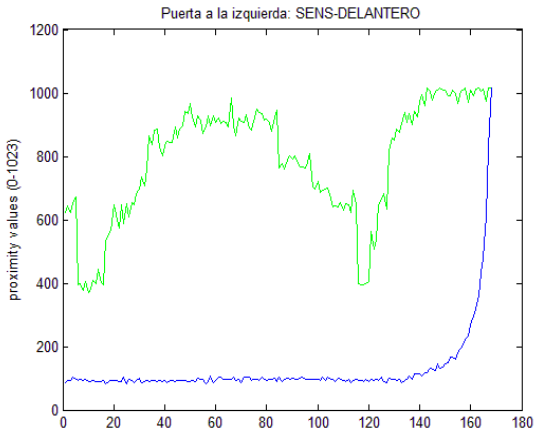


Fig. 29

El simulador KIKS permite elegir entre los robots Khepera I y Khepera II. En nuestro caso nuestro robot real se trata del Khepera II, pero probamos el controlador pasillo con el Khepera I. Para ello ha sido necesario ajustar las medidas del laberinto a las medidas del Khepera I.

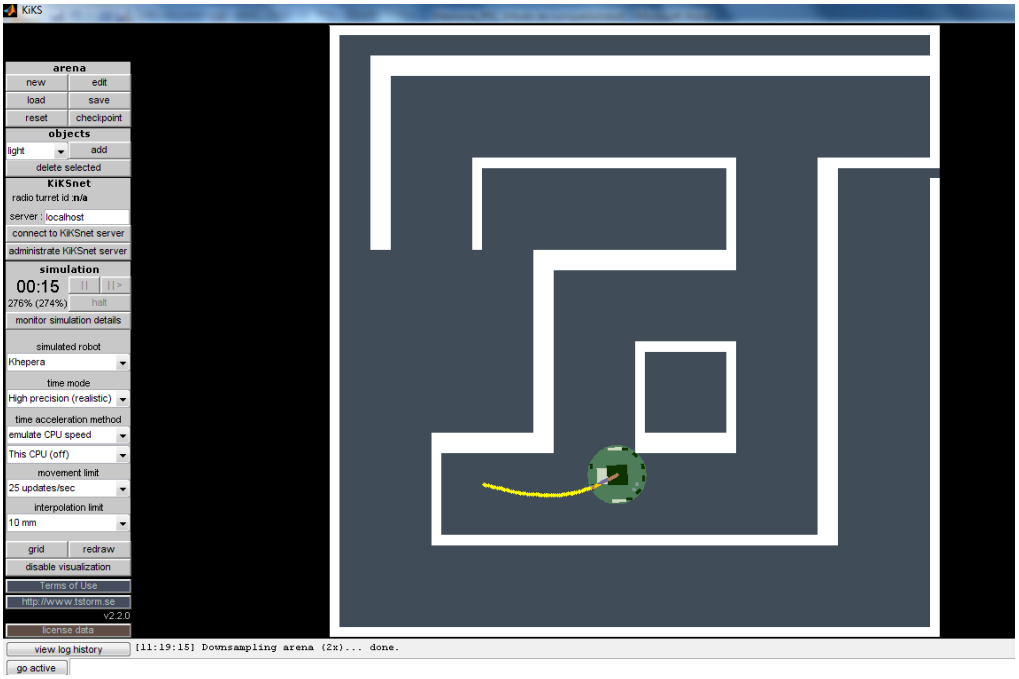


Fig. 30

El funcionamiento del controlador tampoco ha sido satisfactorio en este caso. Para entender la razón de ello observamos las graficas de las mediciones sensoriales realizadas (*ver fig.31).*En ellas se puede detectar que los sensores delanteros en el Khepera I, a diferencia del Khepera II, resultan más realistas. Sin embargo, en el Khepera I simulado tampoco se observa linealidad en los sensores, añadiendo mucha dificultad a la adaptación de sensores.

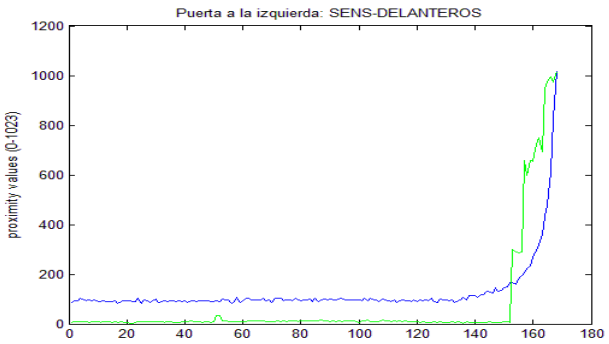


Fig. 31

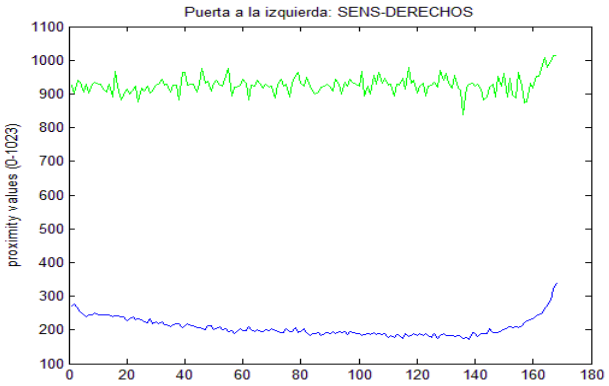


Fig. 32

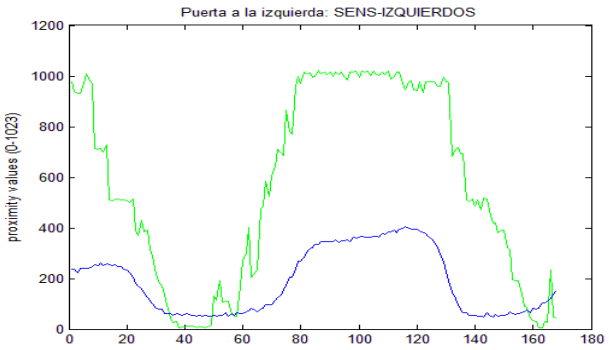


Fig. 33

Como consecuencia de la dificultad de adaptar los sensores se ha tomado la decisión de cambiar de simulador, utilizando otro en el que las medidas de los sensores sean más lineales. El simulador elegido ha sido Webots.

4.1.2. Simulador Webots

4.1.2.1. Introducción

Webots es un simulador avanzado de robótica, trae herramientas para definir modelos propios, definir la física (el entorno o mundo), escribir controladores para los robots y hacer simulaciones a gran velocidad. Existen versiones para Windows, MacOS y Linux, pero la mayor ventaja frente a otros simuladores es sin duda alguna la posibilidad de escribir código en C++, C, Python, Java y Matlab, ya que incorpora API's para estos lenguajes, además las bibliotecas de robot incluidas permiten transferir sus programas de control a muchos robots móviles verdaderos disponibles en el comercio como e-puck, khepera, etc.

Por lo mencionado anteriormente, se podría decir que aunque hemos tenido que cambiar de simulador para continuar con el proyecto, algunas de las características de Webots nos han facilitado la tarea:

- Existe un “bot” implementado para el robot Khepera con su respectiva API de funciones de control, lo cual es indudablemente el primer factor positivo para elegir este simulador.

- Seguiremos programando en Matlab, por lo que podemos reutilizar el código generado hasta el momento.

- Para ser un simulador en 3D, la interfaz gráfica es sencilla y bastante amigable.

- Es un simulador muy poderoso y con gran documentación disponible.

- Tiene muchas ventajas frente a otros simuladores a la hora de trabajar con técnicas de optimización como los algoritmos genéticos.

4.1.2.2. Primeros pasos con Webots

En primer lugar, para empezar a utilizar el simulador necesitamos recrear el laberinto real como mundo webots (.wbt). Así pues, tras realizar las medidas del laberinto real, tenemos que crear el mapa de puntos necesario a la hora de generar las paredes del mismo, ya que al tratarse de un simulador en tres dimensiones son necesarias las coordenadas (x, y, z) para cada punto del entorno.

El laberinto real es el siguiente:

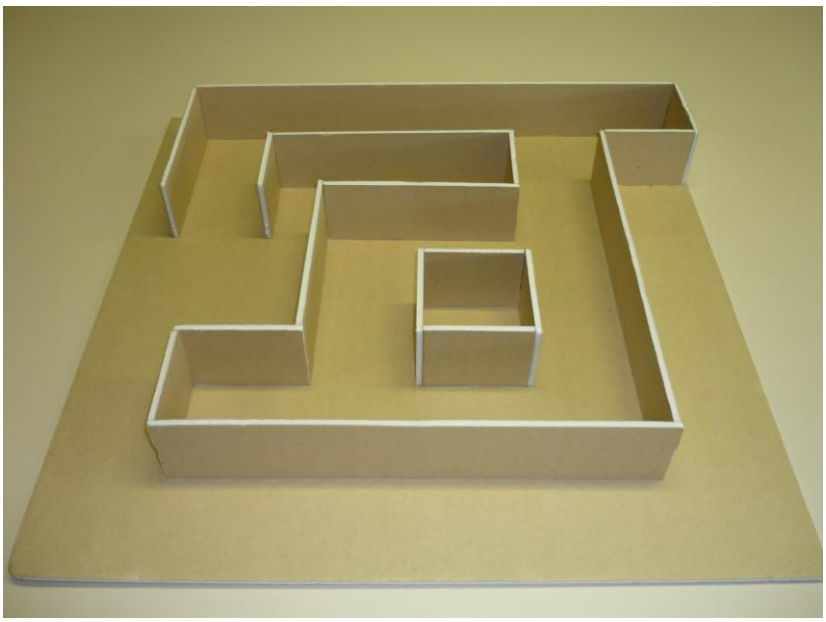


Fig. 34

Así pues, una vez creado el laberinto en el simulador, introducimos un objeto “bot” que será nuestro robot Khepera.

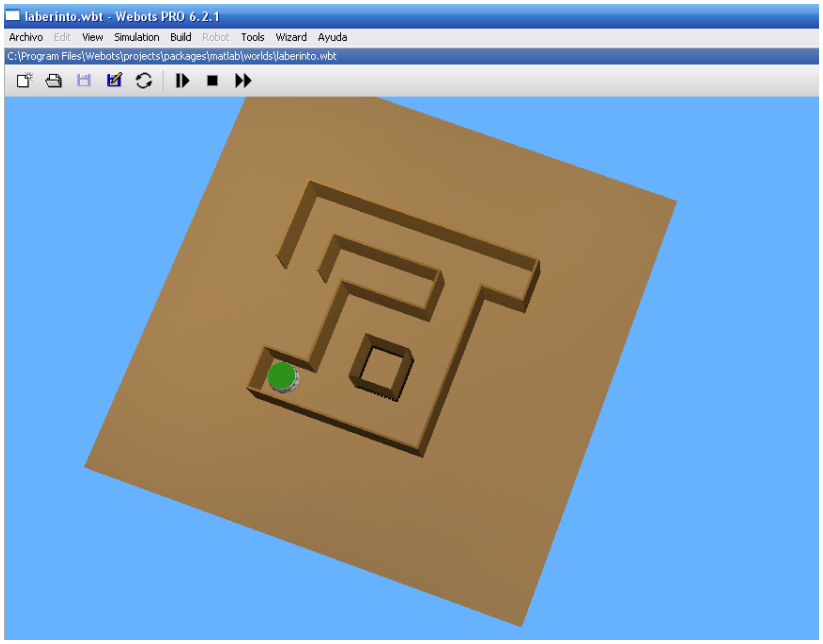


Fig. 35

Ya está todo a punto para poder empezar a trabajar con el simulador, esto es, para interactuar con el “khepera virtual” mediante el controlador previamente diseñado.

4.1.2.3. Adaptación de sensores

Llegados a este punto, con el mundo del laberinto creado, el robot Khepera incorporado, y el controlador para el recorrido implementado (para el otro simulador), es hora de probar el funcionamiento de dicho controlador.

Ponemos en marcha la simulación, pero los resultados no son los esperados, prácticamente no anda ni 2 centímetros y el robot se choca contra una de las paredes. Esto podía pasar, ya que cada simulador tiene sus métodos a la hora de calcular los valores de los sensores; ahora tenemos que adecuar los valores de los sensores del robot a los del nuevo simulador, recordemos que con el simulador Kiks nos fue imposible realizar esta conversión.

En primer lugar, realizamos una serie de pruebas y mediciones, tanto con el robot real como con el nuevo simulador, guardando en ficheros los valores obtenidos por los sensores de proximidad. Tras elegir las mejores mediciones teniendo en cuenta que ambas debían presentar más o menos el mismo número de valores, probamos a realizar ajustes con la herramienta de Matlab “**cftool**” (*Curve Fitting Tool*).

Éstas son las gráficas que muestran los valores del robot y del simulador obtenidos en la medición escogida, sin ajustar:

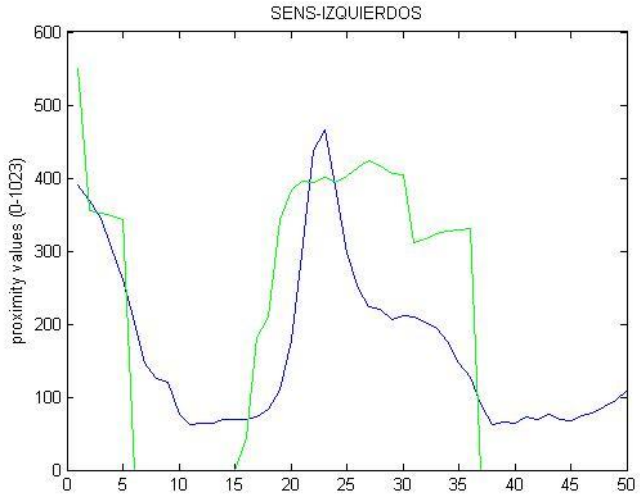


Fig. 36

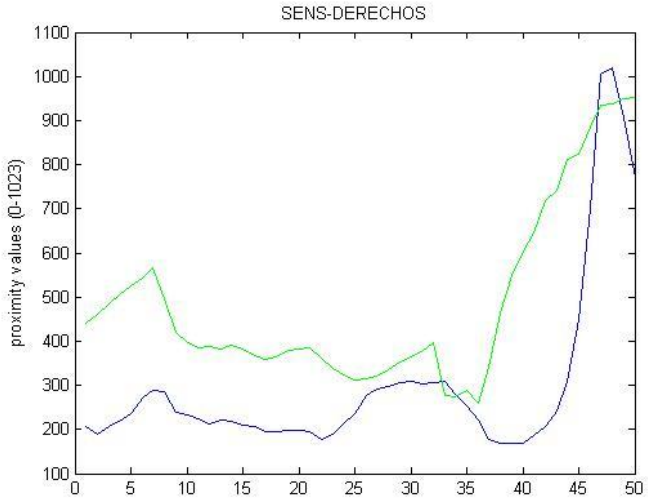


Fig. 37

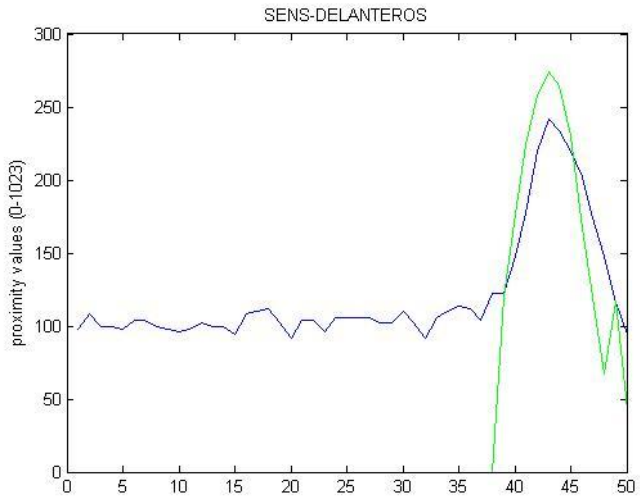


Fig. 38

En la siguiente tabla se muestran las gráficas que muestran el ajuste escogido junto con sus respectivas funciones:

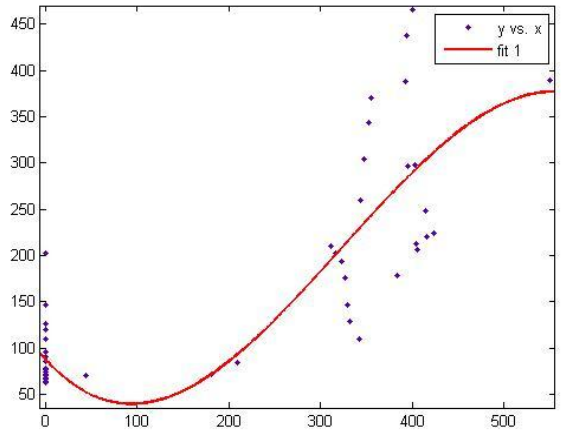
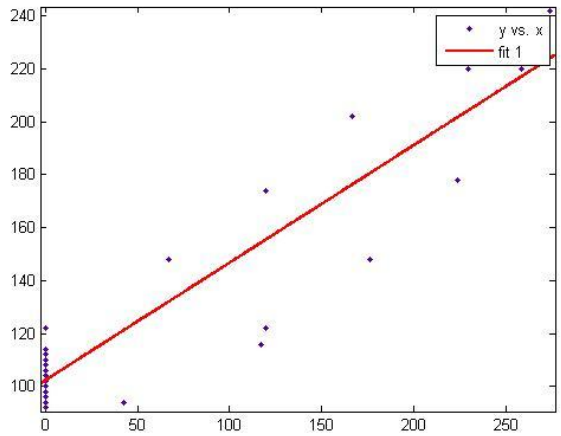
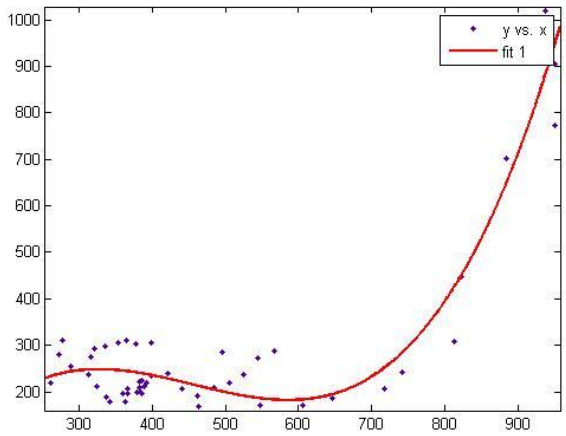
IZQUIERDOS		Linear model Poly2: f(x) = p1*x^2 + p2*x + p3 Coefficients (with 95% confidence bounds): p1 = 0.001109 (7.789e-006, 0.002211) p2 = 0.03529 (-0.4164, 0.487) p3 = 84.45 (57.56, 111.3)
DELANTEROS		Linear model Poly1: f(x) = p1*x + p2 Coefficients (with 95% confidence bounds): p1 = 0.4441 (0.3981, 0.4902) p2 = 102.2 (97.98, 106.5)
DERECHOS		Linear model Poly2: f(x) = p1*x^2 + p2*x + p3 Coefficients (with 95% confidence bounds): p1 = 0.003221 (0.002577, 0.003865) p2 = -3.131 (-3.918, -2.343) p3 = 936.5 (726.6, 1146)

Tabla 26

Otro de los factores clave a comprobar son los valores de las coordenadas (x, y) sobre el laberinto, ya que puede que los “encoders” del

Khepera (contadores de pulsos que nos sirven para poder calcular la posición (x, y) del robot) en el simulador no funcionen igual que los del robot real. Para ello, ponemos en marcha la simulación con el nuevo controlador incorporado, el cual incluye la adaptación de los valores de los sensores, y guardamos ahora los valores de dichas coordenadas.

Comparando ahora las coordenadas obtenidas durante el recorrido del laberinto completo por el robot real, y del recorrido parcial (ya que tras dar la vuelta completa y seguir por el pasillo derecho colisiona) por el simulador, vemos que son muy similares, incluso podemos decir que el simulador tiene menos errores acumulados y es más exacto en lo que respecta a las coordenadas reales. Estas conclusiones pueden deducirse fácilmente tras analizar las siguientes gráficas:

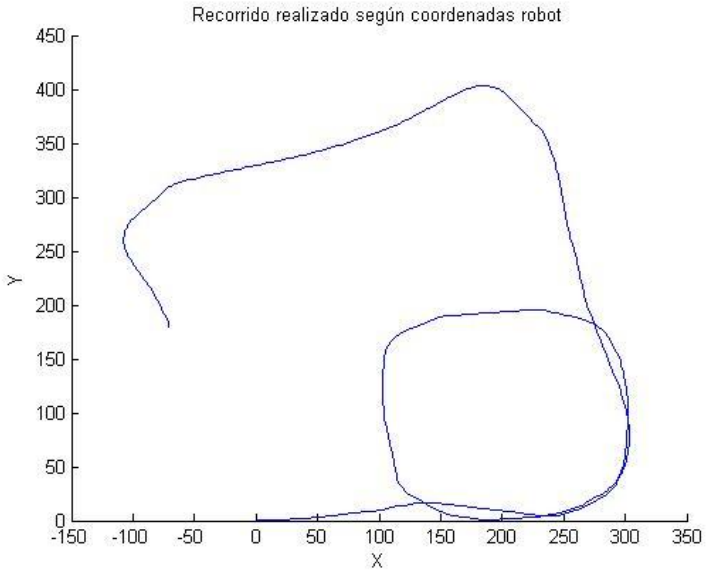


Fig. 39

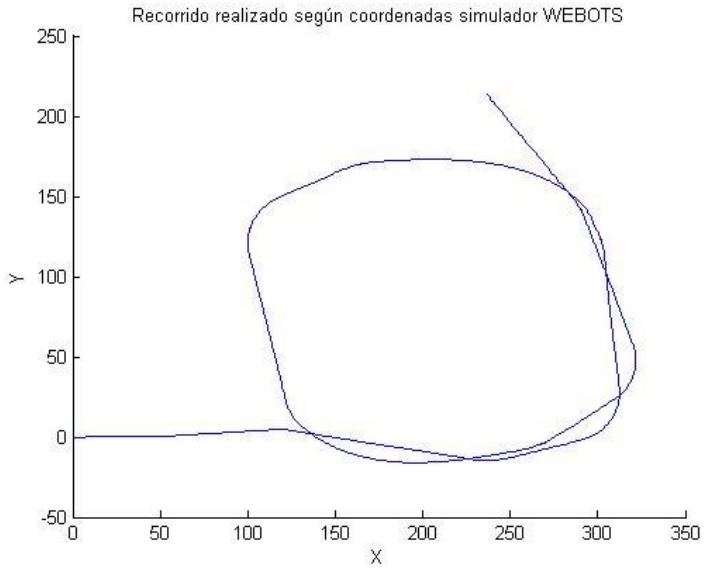


Fig. 40

Así pues, a la hora de indicar en el controlador del recorrido los rangos de coordenadas para entrar por las puertas, debemos modificar los valores, ya que difieren un poco de los que teníamos para el robot real.

4.1.2.4. Controlador normal y supervisor

En esta fase del proyecto, debemos desarrollar el controlador completo para poder realizar varias simulaciones, ya que además del controlador inicial, se necesita otro controlador de tipo “supervisor”.

Un proceso supervisor es similar a un proceso controlador normal. Sin embargo, tiene más capacidades que un simple controlador de robot. Se puede usar para crear un ambiente dinámico (por ejemplo: mover objetos mientras la simulación está en marcha), seguir la trayectoria de los robots, reiniciar la posición inicial de los robots, enviar mensajes a dichos robots, capturar imágenes de las escenas en situaciones determinadas, etc.

Por ello, necesitamos un controlador de este tipo, ya que necesitamos reiniciar o revertir la simulación cada vez que el algoritmo genético llame a la función de evaluación, la cual consiste, lógicamente, en realizar la simulación y devolver un valor que indique “cómo de buena es la simulación”, y por lo tanto el archivo **.fis,** según una serie de factores.

Además, para poder revertir la simulación necesitamos saber cuándo acaba el robot el recorrido, que en caso de colisionar y quedarse atascado será el tiempo máximo indicado como parámetro de la simulación, por lo que hay que intercomunicar el controlador normal con el supervisor para el paso de mensajes. En nuestro proyecto, nos basta con crear una comunicación unidireccional, ya que sólo necesitamos que el Khepera avise al Supervisor. Esta tarea la realizaremos mediante la creación de un emisor en el bot Khepera y un receptor en el Supervisor. Para ello, creamos 2 nodos en la estructura árbol, uno en el Khepera y otro en el Supervisor, indicando en ambos casos el tipo de emisor/receptor “**radio**” y el mismo canal de comunicación, en nuestro caso el canal nº 13.

Fig. 41

4.1.2.5. Comunicación Matlab-Webots vía sockets

Una vez tenemos el controlador normal y el supervisor intercomunicados y tras programar el paso del mensaje por dicha comunicación, ya podemos ejecutar más de una simulación consecutiva, y por lo tanto, empezar a implementar nuestro algoritmo genético.

Sin embargo, aún queda una parte importante relacionada con el funcionamiento del simulador, ya que tenemos que adaptar el código de la librería "**ga_mathworks**", la cual contiene las funciones necesarias para ejecutar el algoritmo genético, al código del simulador, dicho de otra manera, tenemos que hacer que se comuniquen los programas Matlab y Webots. Para ello, mediante el uso de otra librería, en este caso "**msocket**", creamos 2 conexiones vía socket:

Esta conexión sirve básicamente para sincronizar los procesos, ya que hace esperar al algoritmo genético en sí (*genetic.m*) para que no vuelva a llamar a la función de evaluación hasta que ésta no termine con la llamada anterior.

Esta otra conexión se utiliza para mandarle el valor de adaptación del individuo en tratamiento desde el simulador a la propia "función de evaluación" la cual se lo devolverá al algoritmo genético.

4.2. Condiciones iniciales del experimento

Para empezar con la optimización, en primer lugar comenzaremos con el controlador de "seguir pasillo". Una vez optimizado éste, pasaremos a optimizar el otro controlador, el de "entrar por puerta". Dicho esto, cabe destacar que en ambas optimizaciones, debemos escoger bien los valores adecuados para los factores de la función de evaluación, anteriormente explicada en el apartado 5.3.3, y además, debemos elegir un valor apropiado para el factor alfa del operador de cruce. Escoger bien los valores mencionados anteriormente es una tarea algo complicada, por ello tendremos que realizar varias simulaciones antes de dar con la solución ideal del algoritmo genético.

En un principio, con los controladores generados a priori, es decir, con valores aproximados, el controlador completo para el recorrido del laberinto no funcionaba como era de esperar. Siempre colisionaba con una de las esquinas del "pasillo 2" como se puede apreciar en la siguiente imagen:

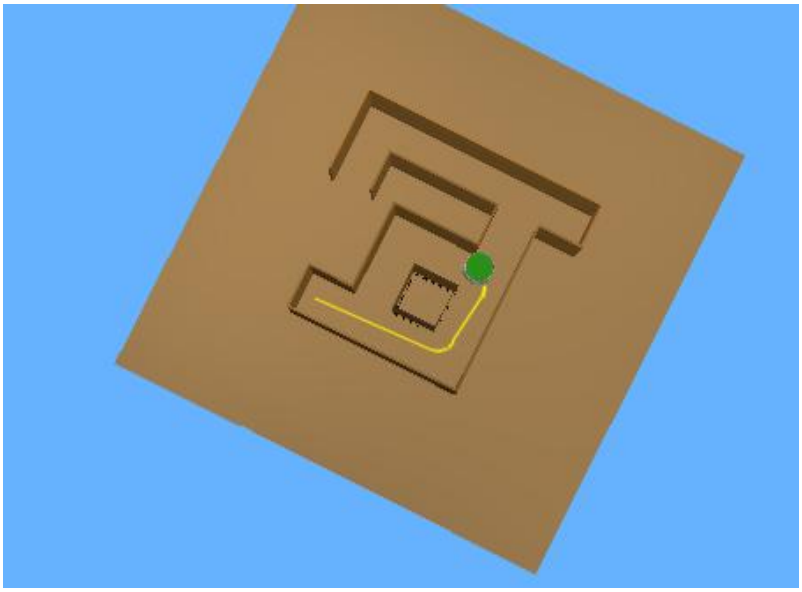


Fig. 42

A lo largo de los siguientes ensayos experimentales esperamos conseguir que el robot realice el recorrido completo del laberinto. Pero no sólo eso, además deberá realizarlo en el menor tiempo posible, recorriendo la menor distancia posible (es decir, no con demasiados giros bruscos innecesarios) y con la menor aproximación posible a las paredes o menor número de roces.

Cabe destacar que el recorrido propuesto sólo contiene maniobras de giro hacia la izquierda así como puertas al lado izquierdo. Esto puede suponer una optimización del controlador buena para este laberinto o recorrido en cuestión pero pésima para otros, ya que a la hora de girar hacia la derecha no sabemos el resultado que nos brindará la optimización obtenida. Esto es, desconocemos si será capaz de realizar el movimiento de giro hacia la derecha.

4.3. Resultados simulaciones

4.3.1. Controlador del pasillo, primera simulación (Controlador 1)

Factores de la función fitness y operador de cruce	
f_colisión	-150
f_roces	-10
f_checkpoints	250
f_distancia	3
f_tiempo	10
(operador cruce) α	0.4

Tabla 27

Los check-points para las 3 simulaciones referentes a la optimización del controlador del pasillo están situados como se muestra en la siguiente imagen:

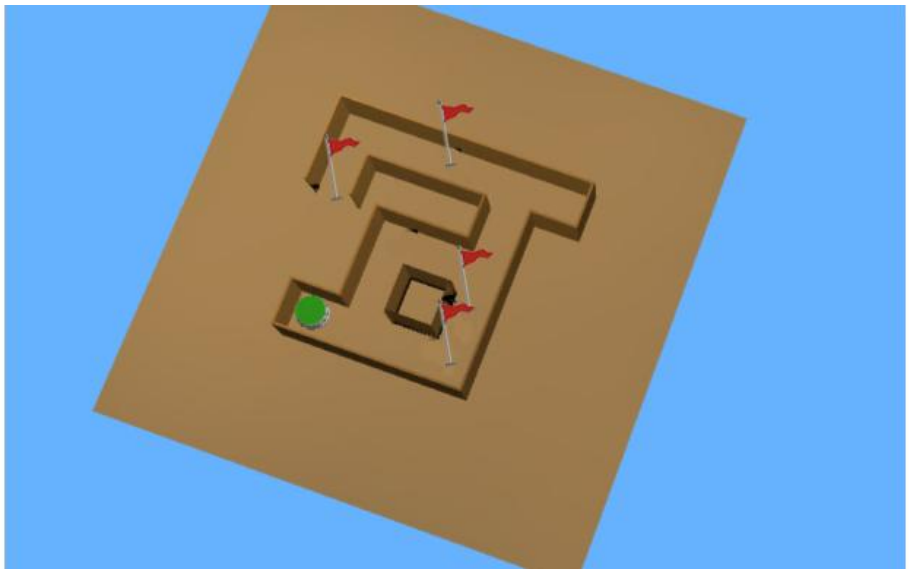


Fig. 43

Resultado obtenido tras aplicar el algoritmo genético

Generation	Maximum	Minimum	Mean	Std. dev.
0	4442.61	-355.335	396.033	1237.82
1	4461.28	-397.575	2407.82	1911.16
2	4461.09	-622.72	3390.42	1596.7
3	4464.35	-278.772	3162.56	1819.76
4	4466.82	144.011	3761.76	1306.41
5	4471.89	172.094	3868.91	1269.87
6	4477.55	-435.56	3454.16	1756.11
7	4477.56	133.422	3783.98	1266.74
8	4477.79	-277.976	3592.28	1523.71
9	4484.36	-305.268	3627.8	1578.66
10	4484.4	-276.397	3659.9	1566.29
11	4485.57	-267.19	3761.23	1563.49
12	4487.37	746.804	4215.76	682.275
13	4487.12	-207.853	3457.76	1780.13
14	4491	1691.51	4214.14	515.244
15	4489.92	-268.704	3593.96	1584.69
16	4491.18	-290.817	3822.62	1326.78
17	4490.89	-218.979	3284.49	1687.45
18	4493.17	-296.358	3387.12	1693.34
19	4493.18	-299.41	2796.04	1997.34
20	4494.56	48.8133	3217.27	1758.04
21	4495.02	-374.485	2808.12	2020.96
22	4496.21	-122.931	3594.35	1585.29
23	4496.52	-242.708	3941.3	1314.46
24	4495.96	-241.507	3880.72	1326.19
25	4495.45	-232.496	3522.84	1759.79
26	4495.25	-317.541	3722.29	1581.06

Genetic algorithm converged.

Tabla 28

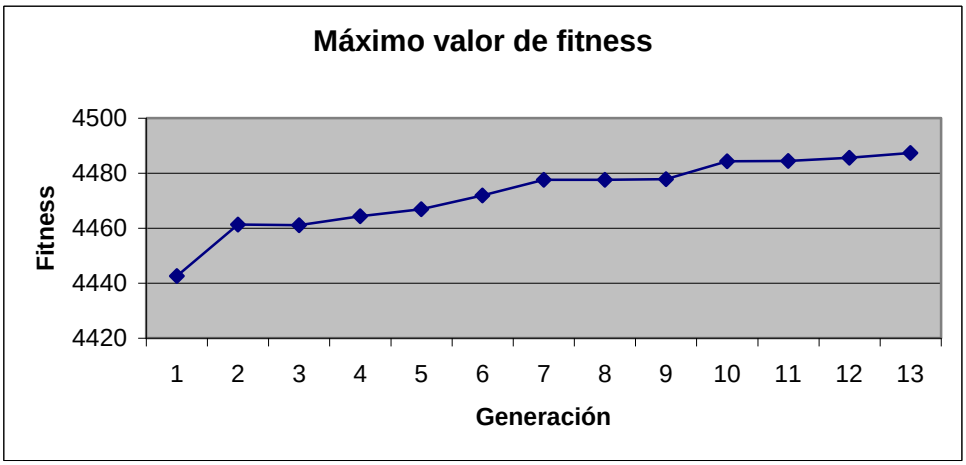


Fig. 44

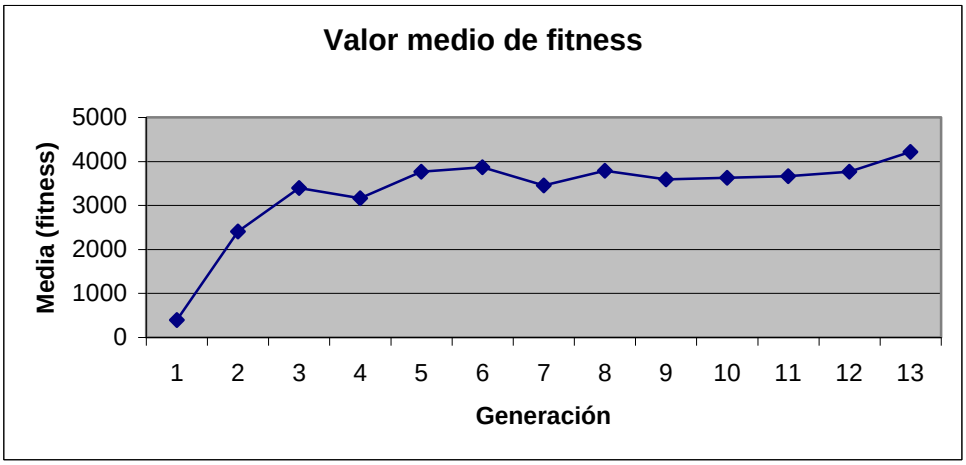


Fig. 45

Valores que toman los factores con el mejor individuo	
colisión	0
roces	1
checkpoints	4/4
distancia	1178,3
tiempo	2,0862

Tabla 29

4.3.2. Controlador del pasillo, segunda simulación (Controlador 2)

Factores de la función fitness y operador de cruce	
f_colisión	-150
f_roces	-10
f_checkpoints	250
f_distancia	2
f_tiempo	10
(operador cruce) α	0.4

Tabla 30

Resultado obtenido tras aplicar el algoritmo genético

Generation	Maximum	Minimum	Mean	Std. dev.
0	3280.47	-389.881	195.42	937.997
1	3300.57	-283.932	2527.29	1170.13
2	3300.91	-292.164	2474.7	1312.56
3	3299.31	192.456	2797.52	944.448
4	3309.33	-43.8814	2750.34	1157.9
5	3305.89	-305.618	2872.73	1039.96
6	3304.88	-261.669	2726.86	1153.06
7	3305.92	-298.146	2538.97	1328.4
8	3309.9	-301.079	2581.52	1396.4
9	3311.72	-286.387	2740.22	1189.77
10	3323.51	-320.56	2635.4	1279.96
11	3322.61	-397.962	2290.25	1506.15
12	3326.5	-21.952	2618.29	1263.41
13	3325.24	-24.3663	2656.14	1235.56
14	3325.04	-318.423	2816.6	1134.03
15	3324.94	-291.764	2487.07	1324.88
16	3325.29	-331.955	2756.58	1181.86
17	3324.7	-462.715	2479.44	1385.49
18	3325.59	-559.484	2572.99	1336.2
19	3326.22	-327.151	2145.48	1519.27
20	3326.18	-357.205	1960.46	1632.06
21	3325.93	-304.383	2598.82	1320.4
22	3325.75	-315.464	2581.22	1311.07
23	3323.99	-302.739	2581.16	1292.12
24	3327.49	-320.885	2442.56	1355.31
25	3325.92	-316.84	2360.24	1461.33
26	3324.11	-322.267	2470.78	1462.26
27	3325.78	-384.943	2914.94	1037.04

Genetic algorithm converged.

Tabla 31

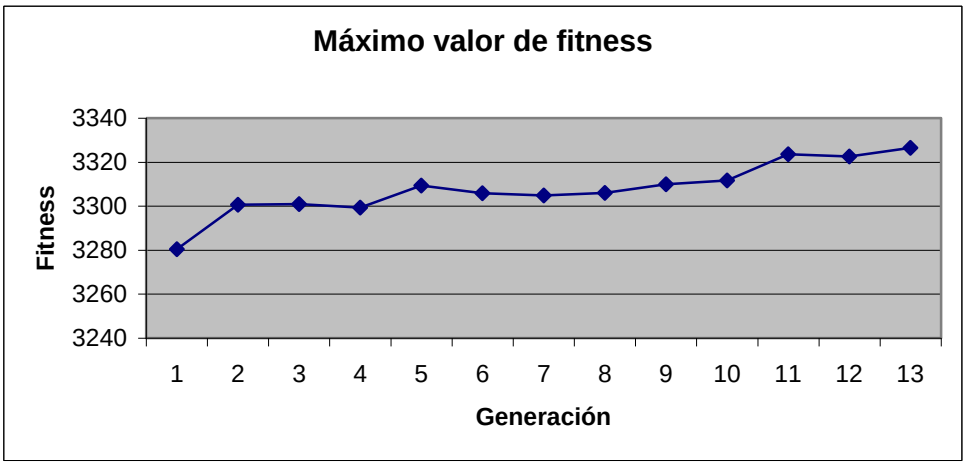


Fig. 46

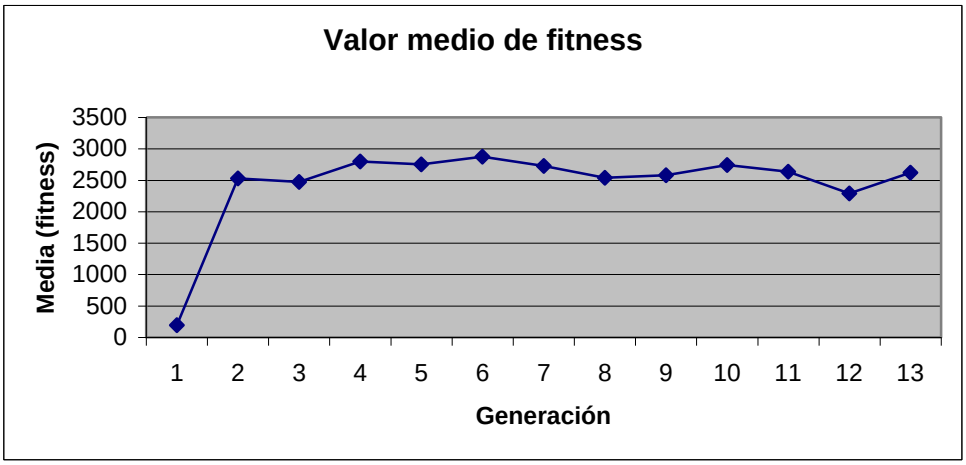


Fig. 47

Valores que toman los factores con el mejor individuo	
colisión	0
roces	0
checkpoints	4/4
distancia	1180,5
tiempo	2,0887

Tabla 32

4.3.3. Controlador del pasillo, tercera simulación (Controlador 3)

Factores de la función fitness y operador de cruce	
f_colisión	-150
f_roces	-10
f_checkpoints	250
f_distancia	2
f_tiempo	10
(operador cruce) α	0.6

Tabla 33

Resultado obtenido tras aplicar el algoritmo genético

Generation	Maximum	Minimum	Mean	Std. dev.
0	3281.77	-497.205	57.7849	890.716
1	3281.41	-413.548	415.118	819.85
2	3298.14	-269.992	1239.1	1349.42
3	3301.76	-397.14	2203.27	1465.88
4	3311.5	-289.985	2617.52	1155.81
5	3313.09	-711.273	2648.15	1225.12
6	3312.41	-210.268	2378.59	1364.77
7	3311.9	-347.22	2098.56	1415.83
8	3312.94	-424.045	2573.5	1249.49
9	3313.16	-206.248	2295.82	1334.55
10	3313.59	-354.957	2530.28	1237.65
11	3314.34	-291.161	2601.99	1214.83
12	3313.79	285.113	2526.26	1072.72
13	3316.64	-294.871	2602.47	1168.86
14	3316.61	-289.131	2884.78	847.267
15	3315.46	-324.441	2454.06	1302.88
16	3322.32	-325.092	2627.24	1228.88
17	3323.54	-309.945	2564.82	1317.8
18	3322.31	-326.88	2529.5	1223.75
19	3323.51	-317.582	2211.12	1553.56
20	3322.37	246.647	2676.8	944.297
21	3323.63	-600.737	2021.24	1584.99
22	3322.35	-108.559	2748.57	1079.97
23	3323.44	-755.13	2156.67	1556.98
24	3323.48	-1030.56	2426.05	1440.96
25	3323.74	-292.206	2598.94	1361.2
26	3323.92	-343.28	2377.75	1351.77

Genetic algorithm converged.

Tabla 34

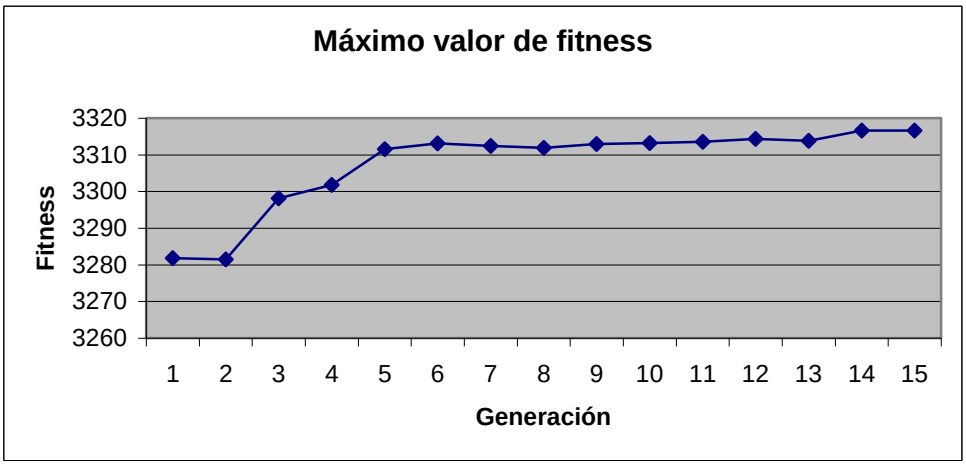


Fig. 48

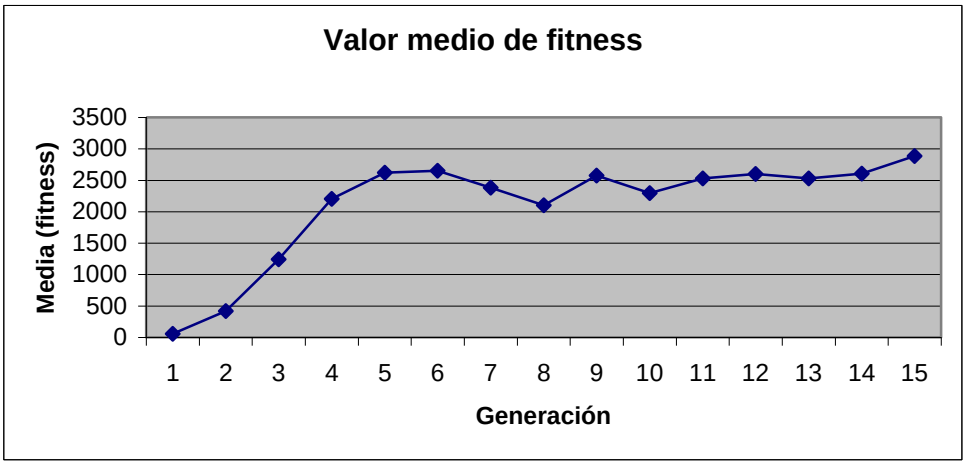


Fig. 49

Valores que toman los factores con el mejor individuo	
colisión	0
roces	0
checkpoints	4/4
distancia	1178,5
tiempo	2,1932

Tabla 35

Dados los resultados anteriores, es decir, partiendo de los datos ofrecidos por el simulador, elegimos como controlador para el pasillo el obtenido en la **segunda simulación**. Sin embargo, en un apartado posterior se podrá ver que éste controlador ofrece peores resultados que otro al usarlo con el robot real, por lo que al final escogeremos ese otro como controlador del pasillo para el controlador definitivo.

4.3.4. Controlador puerta, primera simulación (Controlador 4)

Factores de la función fitness y operador de cruce	
f_colisión	-150
f_roces	-10
f_checkpoints	250
f_distancia	2
f_tiempo	10
(operador cruce) α	0.6
Controlador de pasillo	**Controlador 2 (ver apartado 4.3.2)**

Tabla 36

A diferencia de las anteriores simulaciones, para éstas, es decir, para las correspondientes a la optimización del controlador puerta, los check-points están situados como se muestra a continuación:

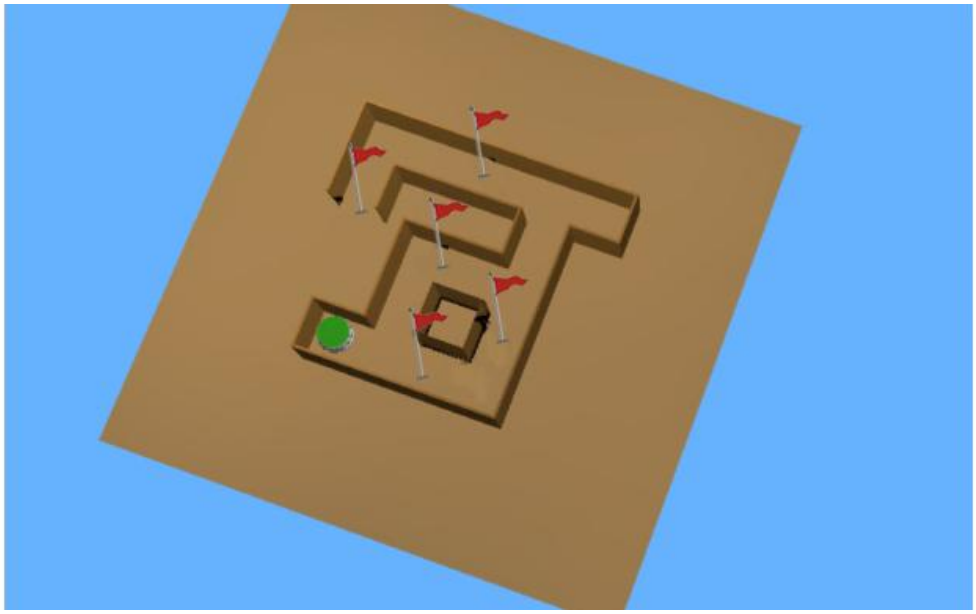


Fig. 50

Resultado obtenido tras aplicar el algoritmo genético

Generation	Maximum	Minimum	Mean	Std. dev.
0	4621.02	1091.94	1338.19	627.246
1	4619.49	968.706	1455.17	709.567
2	4619.5	946.004	1511.66	693.116
3	4635.77	869.618	1794.81	1153.82
4	4635.93	945.475	1975.85	1242.59
5	4635.99	1027.12	2496.13	1418.1
6	4898.88	1134.08	3359.89	1432.46
7	4898.78	1119.82	3015.01	1584.98
8	4898.33	1279.22	3398.94	1538.11
9	4898.6	1271.68	3840.63	1244.58
10	4898.44	1262.23	3580.63	1501.55
11	4893.06	1102.37	3716.04	1250.36
12	4894.95	1166.71	4012.74	1145.38
13	4897.79	1091.02	3838.53	1291.22
14	4895.86	1005.8	3699.88	1486.19
15	4898.23	1035.92	3348.07	1561.56
16	4900.71	803.734	3445.08	1647.8
17	4900.55	207.423	3360.43	1705.18
18	4900.82	1029.71	4039.69	1262.81
19	4898.54	995.807	3495.77	1507.15
20	4906.2	933.135	3313.94	1646.45
21	4901.22	217.32	3210.42	1738.06
22	4902.99	704.642	4118.56	1272.41
23	4904.47	1182.88	3172.53	1614.74
24	4905.41	1113.85	3936.76	1490.52
25	4904.61	1033.99	3996.56	1346.81
26	4906.8	800.455	3233.55	1558.04
27	4910.38	1077.86	3816.63	1398.17
28	4913.64	964.137	3886.39	1420.71
29	4915.83	1094.74	3981.24	1409.59
30	4917.44	777.004	3870.85	1380.31
31	4916.53	988.554	3134.19	1641.45
32	4915.39	1075.49	3171.64	1603.55
33	4915.36	1391.16	4284.18	922.679
34	4915.64	1185.7	4134.07	1089.98
35	4915.87	1081.61	3428.13	1563.72
36	4916.41	1020.69	3400.71	1648.58
37	4915.61	1033.14	3889.66	1415.76

Genetic algorithm converged.

Tabla 37

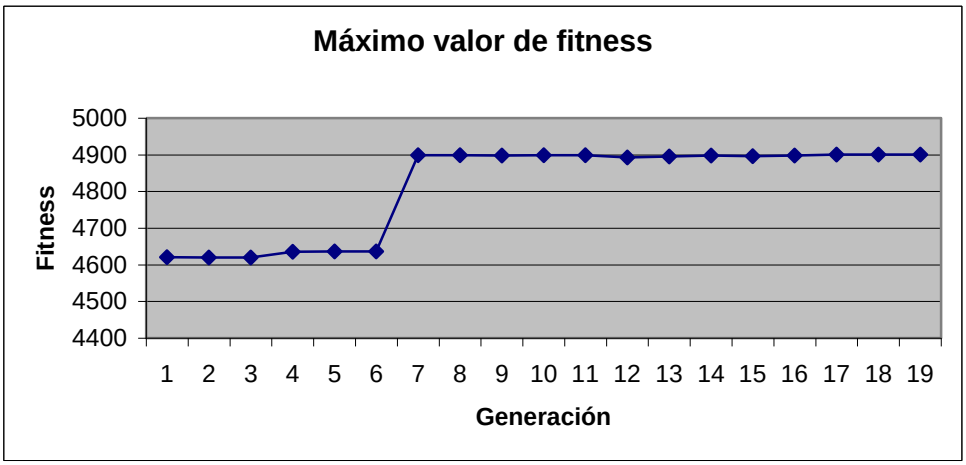


Fig. 51

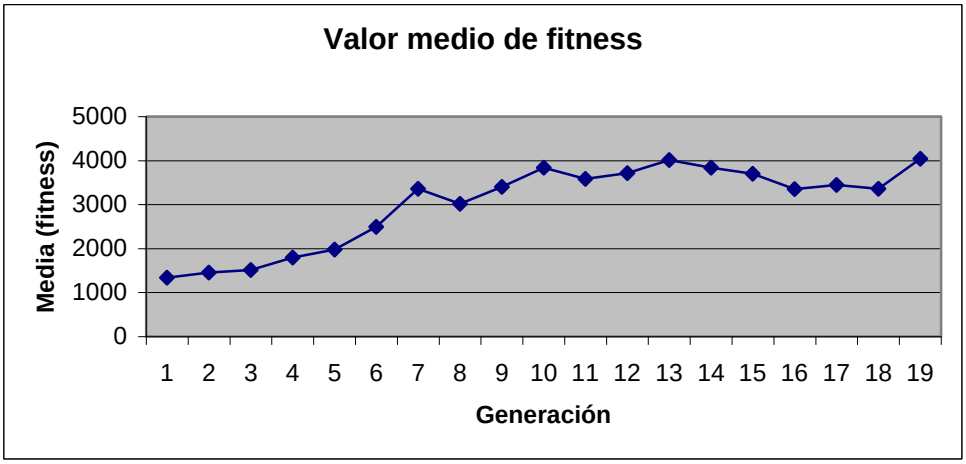


Fig. 52

Valores que toman los factores con el mejor individuo	
colisión	0
roces	0
checkpoints	5/5
distancia	1870,6
tiempo	7,1448

Tabla 38

4.3.5. Controlador puerta, segunda simulación (Controlador 5)

Factores de la función fitness y operador de cruce	
f_colisión	-150
f_roces	-10
f_checkpoints	250
f_distancia	2
f_tiempo	10
(operador cruce) α	0.4
Controlador de pasillo	**Controlador 2 (ver apartado 4.3.2)**

Tabla 39

Resultado obtenido tras aplicar el algoritmo genético

Generation	Maximum	Minimum	Mean	Std. dev.
0	4622	773.099	1376.78	756.232
1	4622.01	939.676	1545.91	962.462
2	4623.2	1011.79	1828.1	1087.57
3	4625.99	1124.8	2097.83	1232.69
4	4630.53	1107.77	2420.36	1538.31
5	4639.07	1139.72	2895.04	1626.82
6	4883.96	1103.22	3097.81	1607.18
7	4883.83	1266.97	3610.14	1400.65
8	4884.42	1139.25	2963.42	1509.33
9	4884.07	1109.47	3200.7	1591.23
10	4884.03	1131.41	3444.26	1536.39
11	4900.06	1116.67	3666.9	1355.12
12	4896.95	1142.39	3549.91	1421.37
13	4897.88	1287.3	3900.39	1088.73
14	4892.24	1296.52	3923.75	1055.51
15	4898.55	1150.53	3482.18	1267.81
16	4898.43	1099.5	3723.09	1258.36
17	4898.24	1383.01	4181.09	880.28
18	4898.45	1135.08	3861.86	1282.47
19	4897.88	1318.9	4113.22	841.288
20	4897.6	1118.2	3787.61	1400.42
21	4898.64	1186.97	3744.65	1159.24

Genetic algorithm converged.

Tabla 40

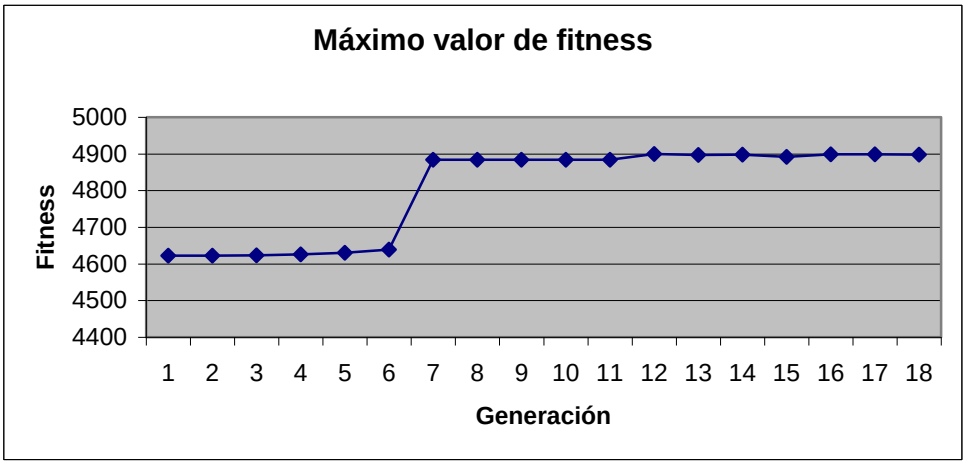


Fig. 53

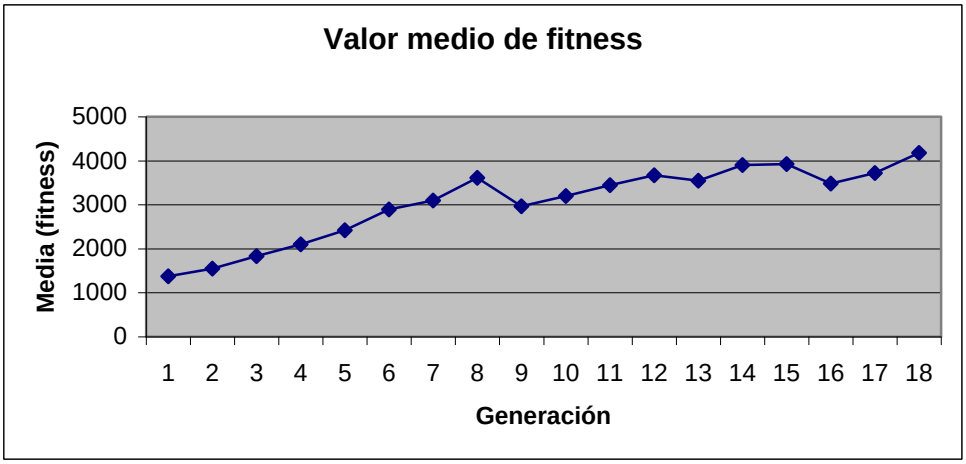


Fig. 54

Valores que toman los factores con el mejor individuo	
colisión	0
roces	0
checkpoints	5/5
distancia	1860,1
tiempo	6,1571

Tabla 41

4.4. Test con el robot real

Una vez optimizados los controladores, en comparación con el que teníamos antes de aplicar el algoritmo genético vemos que los resultados son positivos. Esto se refleja claramente viendo los vídeos del robot real en ambos casos, ya que sin aplicar la optimización éste navega de una forma mucho más brusca y atropelladamente, mientras que el optimizado realiza maniobras más suaves y sutiles. Además, con el controlador inicial el robot no realiza el recorrido si no es desde un punto de partida concreto, ya que hemos tenido que realizar varias pruebas para llegar a verlo funcionar correctamente. Sin embargo, con el controlador optimizado da exactamente igual de dónde parta, porque siempre realiza bien el recorrido.

Una de las conclusiones obtenidas tras obtener el controlador optimizado es que, como ya preveíamos en el apartado inicial del capítulo (*ver apartado 4.1. Condiciones iniciales del experimento*), el robot no es capaz de realizar bien maniobras de giro hacia la derecha. Esta conclusión se obtiene tras poner en marcha el robot desde otra posición inicial, en sentido contrario al recorrido real, veamos el resultado:

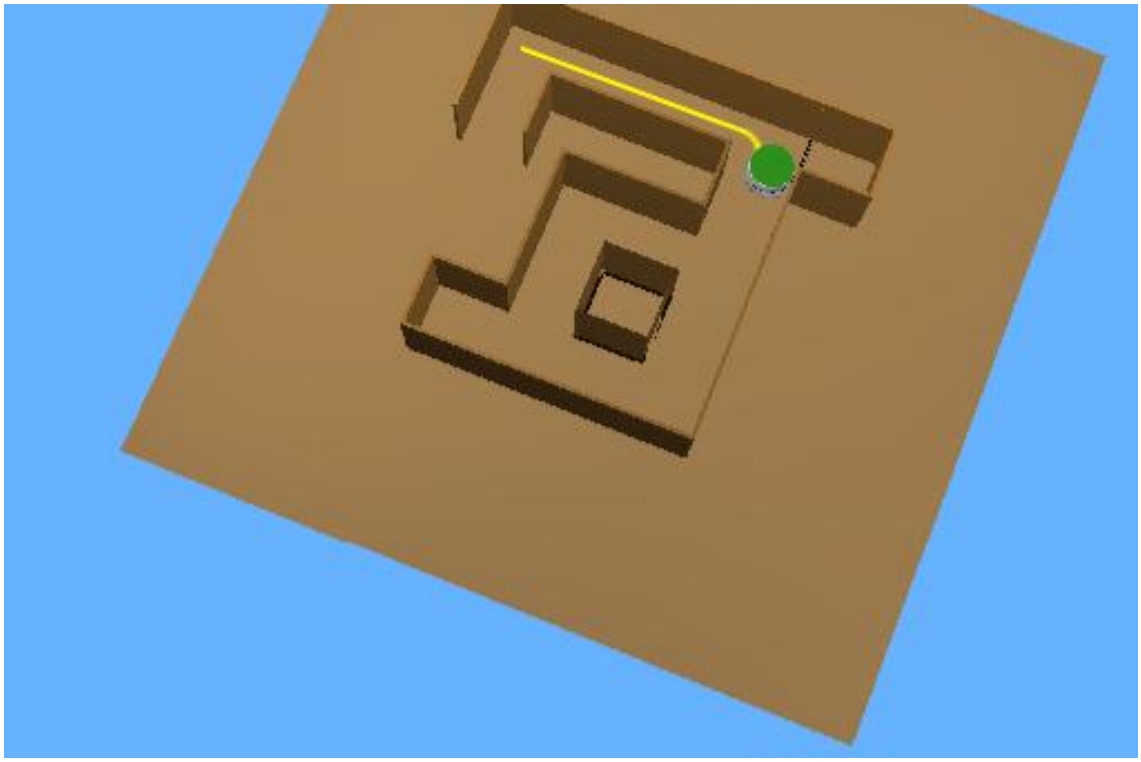


Fig. 55

4.4.1. Elección de controladores según robot real

Basándonos en los resultados obtenidos, no con el simulador sino con el robot real, cabe destacar que el controlador del pasillo utilizado para la optimización del segundo controlador, el de la puerta, ofrece peores resultados que otro de los controladores optimizados:

Resultados obtenidos con el robot real		
CONTROLADOR PASILLO	**TIEMPO**	**INFORMACIÓN EXTRA**
Controlador 1 (primera simulación)	48 seg.	
Controlador 2 (segunda simulación)	¿¿??	Se producen colisiones. No se completa el recorrido con éxito ni una sola vez.
Controlador 3 (tercera simulación)	49,4 seg.	

Tabla 42

Lógicamente, viendo los resultados reflejados en la tabla anterior, para el controlador final elegimos el **Controlador 1** (*ver apartado 4.3.1*). A continuación, tendremos que probar en conjunto éste con el correspondiente para la puerta y elegir el mejor de los dos que habíamos obtenido para el segundo controlador (*ver apartados 4.3.4 y 4.3.5*).

Resultados obtenidos con el robot real		
CONTROLADOR PUERTA	**TIEMPO**	**INFORMACIÓN EXTRA**
Controlador 4 (primera simulación)	¿¿??	Se producen colisiones. No se completa el recorrido con éxito ni una sola vez.
Controlador 5 (segunda simulación)	50 seg.	Los giros no los hace pegados a la pared, va más centrado.

Tabla 43

Una vez realizada esta prueba, cuyos resultados se muestran en la tabla anterior, nos decantamos por el segundo controlador de la puerta, es decir, el **Controlador 5** (*ver apartado 4.3.5*), ya que con el controlador obtenido en la primera simulación no funciona correctamente.

Capítulo 5. Conclusiones

En este proyecto fin de carrera se recoge tanto el estudio y diseño de un controlador difuso para el movimiento autónomo del robot Khepera II, como la optimización del mismo mediante un algoritmo genético adaptado al problema en cuestión.

Para la realización del proyecto comenzamos por diseñar los diferentes controladores difusos con valores aproximados, ya que para crear un controlador general de movimiento eran necesarios varios controladores de comportamientos específicos. Para realizar esta tarea era necesario tener una visión global del patrón de movimiento del robot tanto en el simulador como con el robot real, por lo que se realizaron varios experimentos previos.

Una vez diseñado el controlador completo, hemos utilizado técnicas de computación evolutiva, en nuestro caso un algoritmo genético simple, ayudándonos del simulador Webots.

El resultado tras aplicar la técnica de optimización anterior es un patrón de movimiento mejorado con respecto al que obteníamos con el controlador diseñado a priori (con valores aproximados). Esta mejora del movimiento ha supuesto un menor número de acercamientos a la pared por parte del robot e incluso en ocasiones de roces con la misma. Sin duda alguna la mejora anterior es la que mejor se aprecia y la más valorada, ya que ha hecho que la navegación sea más suave, o mejor dicho, con giros menos bruscos.

En referencia al comportamiento del algoritmo genético, destacar que ha sido mejor de lo esperado, ya que en el primer ensayo experimental ya obtuvimos una mejora considerable. Además, como se puede ver gráficamente en el capítulo anterior (Capítulo 5. Resultados experimentales.), el valor medio del fitness presenta una mejora significativa en cada uno de los experimentos, lo cual nos hace valorar la función de evaluación con nota positiva. En consecuencia, uno de los objetivos más complicados a la hora de diseñar un algoritmo genético, en este caso crear una buena función de evaluación, ha sido completado con éxito.

Respecto al propio algoritmo genético, cabe mencionar que se ha utilizado en conjunción con una política de reemplazo elitista. La aplicación de ésta última ha ayudado a mejorar la eficiencia del mismo, ya que evita que se pierda la mejor o las mejores soluciones. El uso del elitismo se justifica debido a que para mínimos cambios en los genes de un individuo, pueden surgir grandes cambios en el patrón de movimiento del robot, y por tanto puede que se pierda la solución óptima de forma extremadamente rápida. Aunque por contrapartida, también es posible que se hayan descartado individuos que podrían haber evolucionado más rápidamente o mejor.

Algunas de las herramientas de desarrollo utilizadas y los simuladores con los que hemos realizado los ensayos experimentales son de libre distribución y versiones gratuitas o de evaluación. Esto ha causado algún que otro inconveniente en la realización del proyecto, de los cuales el más significativo, sin duda alguna, ha sido la limitación temporal del uso del simulador Webots.

Por último, destacar que a pesar de todos los inconvenientes y dificultades encontradas durante la fase de realización del presente proyecto fin de carrera, hemos conseguido superarlas logrando así incrementar nuestra capacidad para solucionar problemas. Esto es un mérito personal considerable ya que en el mundo de la informática uno tiene que estar preparado para dicha tarea, “solucionar problemas”.

Bibliografía

- [1] Artículo “Mobile robot learning by evolution of fuzzy controller”. Sung-Bae Cho & Seung-Ik Lee, 1998.

- [2] Libro “Redes neuronales y sistemas borrosos”. Bonifacio Martin del Brio y Alfredo Sanz Molina.

- [3] Memoria de un proyecto fin de carrera anterior, “Seguimiento de trayectorias con el robot Khepera II”. Hugo López Gutiérrez.

- [4] www.sc.ehu.es/ccwbayes/docencia/mmcc/docs/temageneticos.pdf

- [5] Página web del simulador Webots. http://www.cyberbotics.com/

- [6] http://www.ayc.unavarra.es/miguel.pagola/

- [7] http://www.depeca.uah.es/docencia/LibreEleccion/IDMRM/trabajos0607/AplicacionesRobotsMoviles.pdf

- [8] http://en.wikipedia.org/wiki/Mobile_robot

- [9] http://omarsanchez.net/Documents/Cinematica_vehículos.ppt